Cazoodle Inc.
60 Hazelwood Drive, Suite 122, Champaign, IL 61820-7460

Web-Scale Search-based Data Extraction and Integration

Final SBIR Phase II Report - October 17th, 2011

Period of Performance: May 2009 - November 2011
SBIR Topic Number: A07-124
Contract Number: W9132V-08-C-0032

Prepared by:
Dr. Kevin C. Chang
Truman Shuck

Principal Investigator:
Dr. Kevin C. Chang
Phone: (217) 265-0299
Email: kevin.chang@cazoodle.com

For:
US Army Topographic Engineering Center
7701 Telegraph Road
Alexandria, VA 22315

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
17-11-2011	Final SBIR Phase II Report	May 2009 - November 2011

4. TITLE AND SUBTITLE
Web-Scale Search-based Data Extraction and Integration: Geospatial Database Generation Agents

5a. CONTRACT NUMBER
W9132V-08-C-0032

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
A07-124

6. AUTHOR(S)
Dr. Kevin Chang, Govind Kabra, Truman Shuck

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
CAZOODLE INC
60 HAZELWOOD DRIVE
CHAMPAIGN IL 61820- 746

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Engineer Research and Development Center CONTRACTING OFFICE
ALEXANDRIA OFFICE
7701 TELEGRAPH ROAD
ALEXANDRIA VA 22315- 3864

10. SPONSOR/MONITOR'S ACRONYM(S)
ERDC

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution A: Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
SBIR Phase II Topic No. A07-124

14. ABSTRACT
Report developed under SBIR contract for topic A07-124, Web-Scale Search-based Data Extraction and Integration: Geospatial Database Generation Agents. In the current age of abundant, digitized geographic data, the classic, manual approach to geospatial feature discovery and gazetteer creation is cost-prohibitive. While geographic data has become increasingly prevalent on the open Web, it remains largely unstructured and difficult to study. This, the GeoEngine project, has developed generalizable methods for automatic gazetteer generation based on the ample, but unstructured data on the open Web. GeoEngine solves this problem with a three tiered architecture: automatic data discovery and extraction, machine-based semantic aggregation and human validation. GeoEngine has produced specific, but generalizable solutions in the following areas: sub-city feature discovery in domestic and foreign locales; neighborhood boundary discovery and refinement; physical feature gazetteer generation and attribute addition; Wikipedia traversal, extraction and auto-correction; and a comprehensive ``Places Profile" of Afghanistan. These methods allow for fast, automated gazetteer generation and support for geospatial research by leveraging the abundance of unstructured data on the open Web and provides new ways of thinking about old problems in geographic information systems.

15. SUBJECT TERMS
SBIR Report; Data Extraction; Data Aggregation; Geospatial Database, Unclassified, Neighborhood Boundaries, Mosque Extraction, Mountain Tagging, Wikipedia, Afghanistan, Populated Places, Gazetteer Generation, Hospital Discovery

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Kevin Chang
U	U	U	UU	165	19b. TELEPHONE NUMBER *(Include area code)* (217) 864-8378

Contents

List of Figures

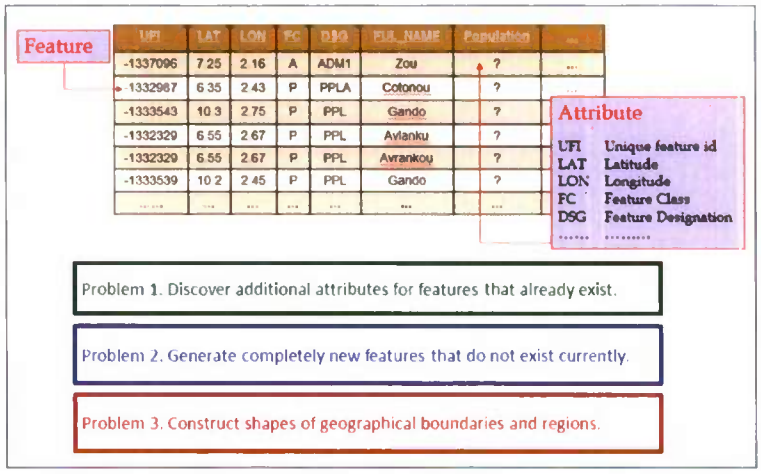


Figure 1: The goal of our SBIR Phase II Project is to enrich geospatial databases.

1 Introduction

1.1 Abstract

Report developed under SBIR contract for topic A07-124, Web-Scale Search-based Data Extraction and Integration: Geospatial Database Generation Agents. In the current age of abundant, digitized geographic data, the classic, manual approach to geospatial feature discovery and geospatial data creation is cost-prohibitive. While geographic data has become increasingly prevalent on the open Web, it remains largely unstructured and difficult to study. This, the GeoEngine project, has developed generalizable methods for automatic geospatial data generation based on the ample, but unstructured data on the open Web. GeoEngine solves this problem with a three tiered architecture: automatic data discovery and extraction, machine-based semantic aggregation and human validation. GeoEngine has produced specific, but generalizable solutions in the following areas: sub-city feature discovery in domestic and foreign locales; neighborhood boundary discovery and refinement; physical feature gazetteer generation and attribute addition; Wikipedia traversal, extraction and auto-correction; and a comprehensive "Places Profile" of Afghanistan. These methods allow for fast, automated geospatial data generation and support for geospatial research by leveraging the abundance of unstructured data on the open Web and provides new ways of thinking about old problems in geographic information systems.

1.2 Background

The goal of our SBIR project was to develop a *GeoEngine* system which could enhance the capabilities of geospatial databases by using information available on the open Web. Consider, for example, a geospatial database as shown in Figure 1. Each row in this database represents a geospatial feature (*e.g.*, a city, a river, *etc.*); each column in the database represents an attribute of the corresponding feature (*e.g.*, name, latitude, longitude, population, *etc.*). Our objective in this project was to develop SBIR Phase I and Phase II technologies for enhancing this database in three aspects:

Problem 1. Discovering missing attributes: Phase I + Phase II As Phase I objective, we aimed at discovering values for additional attributes for features that already exist in a geospatial database. As our target application, we studied the problem of adding the population attribute to populated place features in Benin, Africa. In Phase II, we studied new dimensions of this problem by adding attribute data to mountains in Afghanistan as well as examining and enhancing the attributes of *Wikipedia* articles.

Problem 2. Generating new features: Phase I + Phase II Further, we studied the problem of generating new features by using information available on the open Web. We first developed our techniques for the problem of finding hospital features in Chicago in Phase I, as well as places of worship in Afghanistan, among other features, in Phase II.

Problem 3. Constructing shape boundaries: Phase II Going beyond individual points, we also studied the problem of the discovery of complex features, *e.g.*, generating the shape of geographical regions by inferring the area outline from multiple spot reports. We used open source information to generate the boundaries of neighborhoods in Chicago.

1.2.1 The Core *GeoEngine*: Multi-layer, Multi-extractor Platform

We built the *GeoEngine* system using a suite of technologies which are combined in a *multi-layer, multi-extractor architecture.* The architecture, as shown in Figure 2, is composed of multiple layers of technologies and is designed to aggregate data from sources in a variety of formats.

Layer 1. Geo-Feature Extractor (*DataFactory*) As the first layer, we built an array of data extraction technologies which together provide a "factory" of data access methods. As our key innovation at Cazoodle, our Data Factory is capable of gathering data from "Deep Web" sources, *i.e.*, sources that contain data which are hidden behind query forms or which require complex JavaScript interactions to obtain. Several studies have shown that the data hidden in "Deep Web" sources is far greater in magnitude than data available in the "Surface Web," *i.e.*, the part of the Web that is accessible through static URLs. Typically, search engines are capable of gathering content only from the surface Web. Additionally, our Data Factory provides various "API connectors" for gathering content from proprietary datasets that are hidden behind firewalls or which provide API-based access.

Layer 2. Geo-Feature Matcher The next layer provides technologies for resolving various ambiguities in merging geospatial data obtained from various API data sources. While the *DataFactory* layer organizes content from different sources in a structured database format, different sources may refer to the data elements differently, posing challenges for aggregating this information. For example, for the task of finding population for cities in Benin, we were faced with various ambiguities in matching place names. A city could be referred to by various names (*e.g.*, Cotonou, the capital city of Benin, is also known as *Appi, Kotano or Cotanu*). Likewise, different cities could share the same name (*e.g.*, Benin has 12 cities with the name, "*Gando*"). The Geo-Feature Matcher provides technologies for effective matching of geospatial data obtained from different sources.

Layer 3. Operation Console As the top layer of our architecture, we provide an interactive console for an analyst to visualize the results of the *GeoEngine* system. One of the lessons we learned is that the algorithmic results may not be able to completely substitute for human judgment; rather, the automatic techniques may produce ranked lists of results that can be inspected by an operator, who can make final decisions. Therefore, as part of our technical approach, we built technologies to aid in the creation of operation consoles that an analyst can use to rapidly inspect algorithmic results. For example, in our population task, we built a console that displays the top ranked candidates for the population attribute of the input city; for each candidate answer, the console shows the score, a list of evidence and a snippet of text from each source of evidence. The analyst can click on

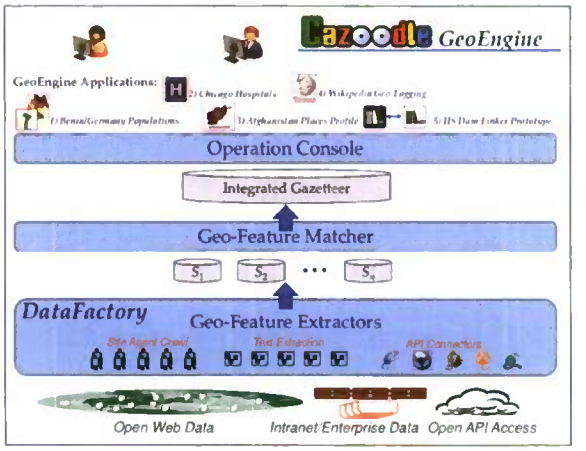


Figure 2: The multi-layer, multi-extractor-based architecture for building the *GeoEngine* system.

any of these pieces of evidence, which will open that source in a preview tab on the right panel. Additionally, the console also includes panels for browsing population results returned by general search engines. The analyst may interactively *drag-and-drop* the most appropriate answer to the bottom-left panel, which will save the results in the output database.

2 Extension of Phase I Work - Hospital Discovery

In the extension of Phase I work, we finished several remaining tasks of our "Hospital Discovery System." In Phase I, we had finished designing of both extraction modules – agent extractor (for structured sources) and text extractor (for unstructured sources). We had also designed the "GeoMerging" algorithm for grouping raw hospital tuples, and applied it to the results from the agent extractors. In Phase II, we continued to accomplish the following remaining tasks.

a) Deployed text extraction to large scale "hospital corpus".

b) Extended the GeoMerging algorithm to text extraction results.

c) Developed scoring functions for ranking the merged hospital records.

d) Combined all components together into an online system.

e) Evaluated performance of the overall system.

2.1 Corpus Extension

For discovering hospital tuples from unstructured text pages, we continued to expand our pilot study of a small set of "hospital-likely" pages to a much larger corpus, to further our techniques. In Phase I, we developed a text extraction module and evaluated its performance using a small corpus of 1000 "hospital-likely" pages, which we obtained from searching "chicago hospitals" at *Microsoft Live* Search. We expanded the experiment from 1000 to 1 million pages.

To begin with, we crawled a large corpus of text pages. We designed a focused crawler program [54, 58] to prepare a large scale corpus of "hospital-likely" pages. Our focused crawler, taking the 1K "hospital-likely" pages as the *seed URLs* to start with, traversed the hyperlinks around them to collect pages from their neighborhood, which are also likely to be pages containing hospital information. Specifically, we feed our crawler with the 1K seed URLs. The focused crawler then started at each of these URLs, parsed the hyper-links on the page to discover new URLs, and continue this "crawling" to gather pages within "depth 3"—*i.e.*, three links away from the seed URL. This focused crawling from the 1000 seed URLs resulted in about 1 million pages overall.

Further, we applied our text extraction module over this corpus of 1M pages so that our text extraction module would skip those pages that have more than *three* addresses. Also, we removed the hospital tuples from text extraction with a score—*i.e.*, the frequency of occurrence from various pages— less than a threshold, which we chose as 10. The low scores of these candidate tuples indicated that they were likely to be incorrect associations of hospital name and addresses. Overall, from this 1M corpus, we obtained 8329 raw hospital records with addresses within 50 miles from Chicago.

2.2 GeoMerging Extension

Our two extraction modules—*i.e.*, Agent Extractor and Text Extractor— produce raw hospital records. Upon these raw records, we need to apply the step of *GeoMerging* to group the raw records that refer to the same hospital feature into a merged record. Our GeoMerging algorithm attempts to address various ambiguity challenges:

- For name: The name of a hospital is not a *unique* identifier—a unique hospital can have multiple names (e.g., "Rehabilitation Institute of Chicago", or "Rehab Center", or "Resurrection Medical Center"), and different hospitals can all have the same names (e.g., St. Joseph Hospital has branches in many cities in USA).

- For address: The same hospital can be referenced using different addresses (e.g., "450 Northwest Hwy, Barrington, IL 60010, USA" vs. "450 W. Highway 22 Barrington, IL 60010"). This may reflect multiple buildings associated with the same hospital.

SampleId	Records Merged	Correctly Merged	Precision	Missed to Merge	Recall
S1	13	13	100%	20	39.4%
S2	26	26	100%	0	100%
S3	11	11	100%	0	100%
S4	1	1	100%	2	33.3%
S5	6	6	100%	0	100%
Overall	57	57	100%	22	72.2%

Figure 3: GeoMerging performance evaluation on text results.

- For phone: The same hospital can appear with different phone numbers that represent different departments in the same building.

For agent-extractor results from structured sources, our GeoMerging algorithm overcomes these challenges using a two-step matching process. 1) *Direct matching*: We merge records when their address, phone, or geo-coordinate attributes are the same. Each of these fields represents a unique way we refer to a "business" and their exact match indicates records can be immediately merged together. 2) *Inferred Matching*: We merge records when their name attribute matches approximately, and at least one of the three attributes—address, phone, and geo-coordinates— matches approximately.

However, does the same scheme work for text-extraction results, which are inherently more "noisy"? While the GeoMerging algorithm worked quite well on agent-extraction results, we found its performance unsatisfactory for text-extraction results. For agent results, the precision and recall were 99.39% and 88%. The performance of the same scheme over text sources was quite poor – because the direct matching step relies on the association of address and names being correct, which does not necessarily hold true for text-based extraction. In the above scheme, the direct matching step simply merges records with the same addresses. Unfortunately, for Text Extractor, the accuracy of name-address association in candidate records is only 15%, *i.e.*, 85% of the tuples potentially represent noisy associations that should not be trusted. Consequently, the direct matching step tends to merge hospital tuples with widely different names, simply due to their same addresses. That is, essentially, errors would *propagate* from extraction of records to their merging.

To accommodate the inherent inaccuracy of Text Extractor in association of name with address, we extended the direct matching step to also look for similarity in hospital name. With this change, our direct matching would not make the mistake of merging raw records that share the same address, when their names are quite different.

We assessed the modified GeoMerging algorithm using the precision and recall metrics. Applying the GeoMerging algorithm on the 8329 text records extracted from the 1M hospital corpus, we obtained 688 merged records with an address within 50 miles of Chicago, IL. To assess these results, we randomly sampled 5 merged records. To evaluate precision, for each sampled merged record, we inspected each of the corresponding raw records, to ascertain how many actually refer to the merged hospital feature. To evaluate recall, for each sampled merged record, we inspected all the raw records that shared the "key term" of the corresponding hospital name, and counted how many our algorithm missed to merge. For example, for the first sampled record, our algorithm merged 13 raw records. Of these, all 13 correctly referred to the merged hospital feature and, therefore, the precision for this sample point is 100%. For recall, for the same sampled record, we manual inspection found that the algorithm missed to merge 20 other raw records—thus, its recall is 13/33 or 39.4%.

Overall, as Figure 3 shows, our algorithm's precision is 100% and recall is 72%. In comparison, the precision and recall of the GeoMerging algorithm on agent-extraction results were 99.4% and 88%, respectively. While the precision metrics of GeoMerging on both results are quite high, the recall for text-extraction results is much lower compared to that of the agent results. Our analysis showed that this recall degradation is due to our modification to the direct matching step—for text results, direct matching also requires hospital names to be similar—which resulted in the decreased recall since a hospital feature can have quite different names (*e.g.*, "resurrection center" and "rehab medical institute")—such place name matching, as well-known in GIS research [63, 64], is inherently imperfect.

We observed, from the *frequencies* versus the *counts* of records, a classic Zipf [102] (or power law) distribution with a "long tail" phenomenon [43]. First, in Figure 4, we show the distribution of the number of raw records per merged record. Of the 688 merged records obtained, we observed a "long tail" distribution. Some sources have very high frequencies of occurrences; *e.g.*, 53 out of 688 were the results of merging 20+ (more than 20) raw

Raw Records per Merged Record	Number of Merged Records
1	362
2	100
3	44
4	29
5	20
6	19
7	10
8	8
9	4
10	8
11	6
12	2
13	7
14	6
15	1
17	4
18	2
19	1
20	2
20+	53
Total	688

Figure 4: Distribution for results of text extraction: Number of raw records per merged record.

Number of Sources per merged record	Number of Merged Records
1	475
2	86
3	39
4	28
5	18
6	14
7	12
8	8
9	2
10	2
11	1
12	1
14	1
21	1
Total	688

Figure 5: Distribution for results of Text Extraction: Number of Sources contributing to each Merged Record.

records—indicating a few popular hospital features are discovered very frequently. On the other hand, however, many records have low frequencies. In particular, 362 records have only one source record (thus, for these "merged" records, there is no merging with others beyond themselves) and 100 have two records. Second, we can similarly observe the long tail distribution for where—how many sources—we can expect to find a record. As Figure 5 shows, we also studied the distribution of the number of sources contributing to each merged record; *i.e.*, the number of sources where a merged record occurs. As we see, the distribution shows a similar long tail distribution.

The observation of the long tail distribution indicates the need for a comprehensive corpus as sources in discovery. Where can we expect to discover a hospital feature? For a few popular hospitals, as they are frequently

Raw Records per Merged Record	Number of Merged Records
1	776
2	229
3	90
4	70
5	58
6	27
7	19
8	14
9	5
10	12
11	10
12	4
13	6
14	8
15	2
16	2
17	2
18	4
19	1
21	7
22	3
23	1
24	2
25+	79
Total	1431

Figure 6: Distribution for combined results: Number of raw records per merged record.

mentioned (indicated from their high frequencies in Figure 5), we can expect to find them even in just a small number of selected sources. However, for the "long tail" of less frequent hospitals, we will need to look for them "everywhere," since they may not appear in the selected sources. Thus, for comprehensive discovery, we will need a comprehensive set of pages to cover as many hospitals as there exist.

2.3 Extraction Results Merging

With both the agent and text results ready, we applied the GeoMerging algorithm on the combined results—the set of 11951 raw records obtained from the agent extractor and the text extractor. After merging, we obtained 1431 hospital features within 50 miles of Chicago, IL. We make two observations.

First, we again see the long tail phenomenon. Similar to the analysis in Section 2.2, for the combined results, we show the distribution of the number of raw records per merged record in Figure 6 and the number of sources per merged record in Figure 7. Evidently, the distributions are of the Zipf-type, where we see a long tail of many hospitals that could be found only in a smaller number of sources. Thus, for comprehensive discovery, we cannot simply focus on a few selected sources; instead, we must be comprehensive over many sources.

Second, to understand the influence of each extractor, we study the "joint" distribution of results from Agent Extractor and from Text Extractor. As Figure 8 summarizes, we analyze the number of text sources and the number of agent sources contributing to each merged record. The columns (x) and rows (y), respectively, represent the number of agent sources and the number of text sources contributing to a merged record. A cell (x, y) records the "count" (*i.e.*, number) of merged records that are discovered by x agent and y text sources. *E.g.*, the cell (2, 3) has a count 5; *i.e.*, there are 5 merged records that were obtained from 2 agent and 3 text sources. Note that, in the table, we use $*$ to represent *any* value of x (or y); *e.g.*, cell (2, $*$) has count 166, which indicates 166 merged records were found in 2 agent sources and any number (0 or more) of the text sources.

The observations from the joint distribution show that the dual extractors—Agent Extractor and Text Ex-

Number of Sources per Merged Record	Number of Merged Records
1	1135
2	122
3	31
4	25
5	13
6	13
7	11
8	7
9	7
10	7
11	9
12	3
13	4
14	14
15	6
16	4
17	9
19	4
20	2
22	2
24	1
27	1
31	1
Total	1431

Figure 7: Distribution for combined results: Number of sources contributing to each merged record.

y: Number of Text Sources	*x: Number of Agent Sources*											
	0	1	2	3	4	5	6	7	8	9	10+	$*$
0	0	447	150	54	40	33	10	8	6	4	25	777
1	329	4	2	4	3	0	0	3	0	1	1	347
2	76	0	4	4	0	2	0	0	0	1	10	97
3	33	2	5	1	0	0	0	0	0	0	2	43
4	21	1	2	0	0	0	0	0	0	0	2	26
5+	85	0	3	1	2	0	1	0	2	2	45	141
$*$	544	454	166	64	45	35	11	11	8	8	85	1431

Figure 8: Joint distribution of the number of agent (x) and text (y) sources contributing to a merged record.

tractor—complement each other well. On the one hand, a fairly large number of merged records were discovered only from agent sources, or only from text sources—indicating that both extractors are essential and they will *complement* each other. The count of cell (0, $*$) shows that as many as 544 of the 1431 merged records were obtained only from the text sources. Likewise, the cell ($*$, 0) counts 777 of the 1431 were obtained only from the agent sources. Thus, both the extractors have unique contributions in our discovery process.

On the other hand, a significant number of merged records were obtained from both types of sources—thus, the two extractors can *reinforce* each other as well. From Figure 7, subtracting 544 of (0, $*$) and 777 of ($*$, 0) from the total 1431, we obtained 110 records which appeared from both extractors. Furthermore, from cell (10+, 5+), we see that 45 merged records are obtained from a large number of agent as well as a large number of text sources. We believe this "diversity" of supporting evidence leads to two implications of our dual-extraction approach:

- *The abundance of evidences across different extractors reinforces the confidence of discovery.* These candidate

1 Northwestern Memorial Hospital
221 E Huron St, Chicago, IL 60611-2957, US
(312) 440-0709
Score = 6560 from 49 A-sources, 1126 T-sources

Name	Address	Phone	Support	Url
Northwestern Memorial Hospital: Northwestern Memorial Foundation	676 N Saint Clair St, #2050, Chicago, IL 60611-2942, US	(312) 926-2033	7	maps.google.com(agent)
Northwestern Memorial Hospital: Crisis Intervention Hotline	251 E Huron St, Chicago, IL 60611-2908, US	(312) 926-9586	7	maps.google.com(agent)
Rehab Institute Of Chicago	345 East Superior Street, Chicago, Illinois, 60611 4496 USA		7	www.hospitalsworldwide.com(agent)
Rehabilitation Institute of Chicago	345 East Superior Street, Chicago, Illinois, IL 60611 USA		7	www.hospitalsworldwide.com(agent)
The Rehab. Institute Of Chicago	345 E. Superior Chicago, IL 60611		7	www.yimo.com (agent)
Rehabilitation Institute of Chicago	345 E Superior St, Chicago, IL	(312) 238-1000	7	local.yahoo.com (agent)
Northwestern-Anesthsia Department	333 E Superior St, #466, Chicago, IL	(312) 926-7632	7	local.yahoo.com (agent)
Keith, Louis G MD - Northwestern Memorial Hospital	333 E Superior St, #464, Chicago, IL	(312) 926-2000	7	local.yahoo.com (agent)
northwestern memorial hospital	345 E Superior St, Chicago, IL 60611-2654, US		1	events.nbcchicago.com(text)
northwestern memorial hospital	345 E Superior St, Chicago, IL 60611-2654, US		1	www.nic.org(text)

Figure 9: The summary snippet of an example merged result.

records are likely to be correct ones. As Section 2.4 will present, we use such repeated evidences as voting in our scoring function for ranking discovered records.

- *The variety of information from different sources enriches the discovery.* While structured sources, through Agent Extractor, tend to give correct results, they are also likely to contain only repeating "directory" listings, such as contact information. Text sources, in contrast, while harder to extract, tend to provide more interesting and diverse contents (*e.g.*, descriptions, reviews, comments). Together, the dual types of sources will reinforce to provide not only correct but also rich information.

2.4 Ranking Algorithm

After combining the dual extractors, we next discuss how the merged hospital records are scored to obtain the final ranked results. As the base score, we score each raw record t = (name, address, phone, source) as follows:

- If t is from text extraction: $Score(t) = \log(W_t)$, where W_t is the Web reference of t (*e.g.*, the hit count of searching t at a search engine). That is, we take the logarithmic value of how many times the tuple has been mentioned on the Web, as an indication of its correctness.
- If t is from agent extraction: $Score(t) = 1$. That is, we treat each agent result from structured sources as equally reliable.

Now, we need to compute the scores of a merged record based on the scores of its raw records. As an example, Figure 9 shows a merged result which is obtained from merging 49 agent records and 1126 text records.

The Result Ranker module essentially aggregates, for each merged tuple, the scores of the supporting raw tuples, much like voting, accounting for the reliability of the originating sources. As just explained, each supporting result is of the form t = (name:n, address:a, phone:p, source:s)—*i.e.*, a new feature with name n, address a, and phone p, which is obtained from source s. Let X be the set of extractors x (we currently have two extractors), and T_x be the set of tuples $t = (n, a, p, s)$ that extractor x produces. We merge them by an aggregation of the tuple scores over all the extractors and all the sources. However, as not all results are equally efficacious, the aggregation must differentiate *where* (from which source) a result is collected, and *how* it is processed (by which extractor). To

Method	Total	Discovered	Correct	Precision	Recall
Struetured Sites (15 agents)	22	22	22	100%	100%
Text Crawl (1M pages)	22	19	19	100%	86%
Overall Combined	22	22	22	100%	100%

Figure 10: Evaluation: 34 hospitals sampled from US News Hospital Direetory.

differentiate the appropriate extractor, we weigh the results from extraetor x by a constant faetor α_x. Speeifieally, our implementation sets $\alpha_1 = 1$ for Text Extraetor and $\alpha_2 = 7$ for Agent Extraetor, to refleet the empirieally obtained 15% aeeuraey of Text Extractor compared to 100% aeeuracy of Agent Extractor. Similarly, we weigh eaeh souree s by a eonstant β_s. Without further source information on souree authority, our implementation eurrently sets $\beta_s = 1$ for all sourees.

The overall seore for a eandidate hospital feature (n, a, p) is simply the weighted sum over all its supporting raw records $t = (n, a, p, s)$, which comes from various sources s and is extracted through different extractors x:

$$Score(n, a, p) = \sum_{x \in X} \alpha_x \cdot \left[\sum_{t=(n, a, p, s) \in T_x} \beta_s \cdot Score(t) \right] \quad (1)$$

For instanee, for the example result in Figure 9, the merged record ("Northwestern Memorial Hospital", "251 E Huron Street, Chicago, IL", "(312) 440-0709") has a final seore 6560. As Eq. 1 states, it sums up the scores of the supporting raw reeords: 49 reeords from Agent Extraetor (*e.g.*, the first 8 rows in the summary table) and 1126 reeords from Text Extraetor (*e.g.*, rows 9 and 10). The eolumn labeled "Support" shows the weight values which differentiate the extractors α_x (for Text or Agent Extractors), since we do not use different β_s for the sources.

We note that, by exploiting sueh "supports," we naturally exploit the "redundaney" of the Web, where the same information may appear multiple times. As the key insight, through the redundaney, we expect correct information will be correct "in the same way," while an incidental error will be wrong "in its own way." While not always true, it is intuitive that information that appears multiple times is more likely to be a "consensus" answer, rather than an ineidental error. We thus take the analogy of "voting", in which we score eaeh final result by aggregating the scores of its supporting raw results.

2.5 Performance Evaluation

To evaluate the overall performanee, we assessed how well it could discover hospital features. We sampled 22 hospital features randomly from all the hospitals listed in *US News Hospital Directory*[1] for the Chicago metropolitan area. The results are summarized in Figure 10.

- For the structured sites using 15 agents, for all the 22 sampled features, the top result from our system is always correct. Thus, a precision of 100% and recall of 100%.
- For the unstruetured sourees using 1M pages, of the 22 sampled features, our system discovered 19 of them. For the 19 features that our system diseovered, the top result was always eorreet. Thus, for text sourees our preeision and reeall are 100%, and 86%, respeetively.
- Overall, for the eombined agent and text results, the top result of our system is correet for all the 22 sampled features, *i.e.*, both preeision and reeall of 100%.

2.6 Operations Console

We now deseribe our design of the Operations Console so that an analyst ean inspeet the results generated by our system and determine entries that should be inserted into the final gazetteer. This task will require several operations earried out with the results generated by our system. We use an illustrative scenario to demonstrate the overall proeess, ineluding various editing operations.

[1] online at http://health.usnews.com/features/health/hospital-directory.html.

Figure 11: Target Geography: The operational bounding box around Chicago.

name	NumSources	latitude	longitude	address	phone
northwestern memorial hospital	36	41.8967	-87.62	303 E Chicago Ave, Chicago, IL 60611-3008, US	
Comer Childrens Hospital	33	41.7909	-87.6048	5721 S Maryland Ave, Chicago, IL 60637-1425, US	(773) 702-9200
Cook County Hospital	27	41.874	-87.6741	1901 W Harrison St, Chicago, IL 60612-3714, US	(312) 633-6000
Saints Mary and Elizabeth Medical Center: Saint Mary Campus	23	41.9029	-87.683	2233 W Division St, Chicago, IL 60622-3043, US	(312) 770-2000
Saint Joseph Hospital	22	41.9349	-87.6369	2900 N Lake Shore Dr, Chicago, IL 60657-5640, US	(773) 665-3000
Illinois Masonic Hospital	22	41.9365	-87.6501	836 W Wellington Ave, Chicago, IL 60657-5147, US	(773) 975-1600
Schwab Rehabilitation Hospital	21	41.8625	-87.6957	1401 S California Ave, Chicago, IL 60608-1612, US	(773) 522-2010
West Suburban Medical Center	20	41.8917	-87.7753	3 Erie Ct, Oak Park, IL 60302-2519, US	708-383-6200
Swedish Covenant Hospital	20	41.9749	-87.6988	5145 N California Ave, #370, Chicago, IL 60625-3661, US	(773) 878-8200
Rush University Medical Center	19	41.875	-87.668	1653 W Congress Pky, #622, Chicago, IL 60612-3833, US	(312) 942-5000

Figure 12: The top ranked results in the final output gazetteer.

2.6.1 Browsing Console

The screenshot in Figure 13 shows the first console for browsing results generated by our algorithm.

- The left panel shows the list of all the results that still need to be inspected by the analyst. The search box can be used to search within the candidate hospital results by name. For the query "north," our system returned 9 matching results that were produced by our algorithm.
- The snippet of each result includes the name, address and phone number of the hospital. The snippet also displays the overall score of this merged result, and the number of the raw agent and text records that were merged to produce this result. The last result in the search list is: "Northwestern Medical Faculty," with an address of "680 N Lake Shore Dr, #1000, Chicago, IL 60611-3057, US" and the phone number as "(312) 695-9797." This merged result was produced by combining 114 agent records and 6 text records, with a final aggregate score of 811, as produced by our GeoIntegration ranking function.
- The panel on the right shows the entries already present in the final gazetteer based on the analyst operations.
- The center panel shows the location of hospitals on a map for easy visualization of each result in the search list. The display shows nearby hospitals, as well as hospital results that have been inserted into the final gazetteer.

2.6.2 Deletion Operation

The results produced by our algorithm may not represent meaningful associations. In fact, of the 9 results shown in the screenshot in Figure 13, the analyst determined that 5 results need to be deleted. Overall, after full editing of the 351 results produced by our algorithm (within our target geography), the analyst deleted 63 results.

We provide the following functions to support the deletion of results.

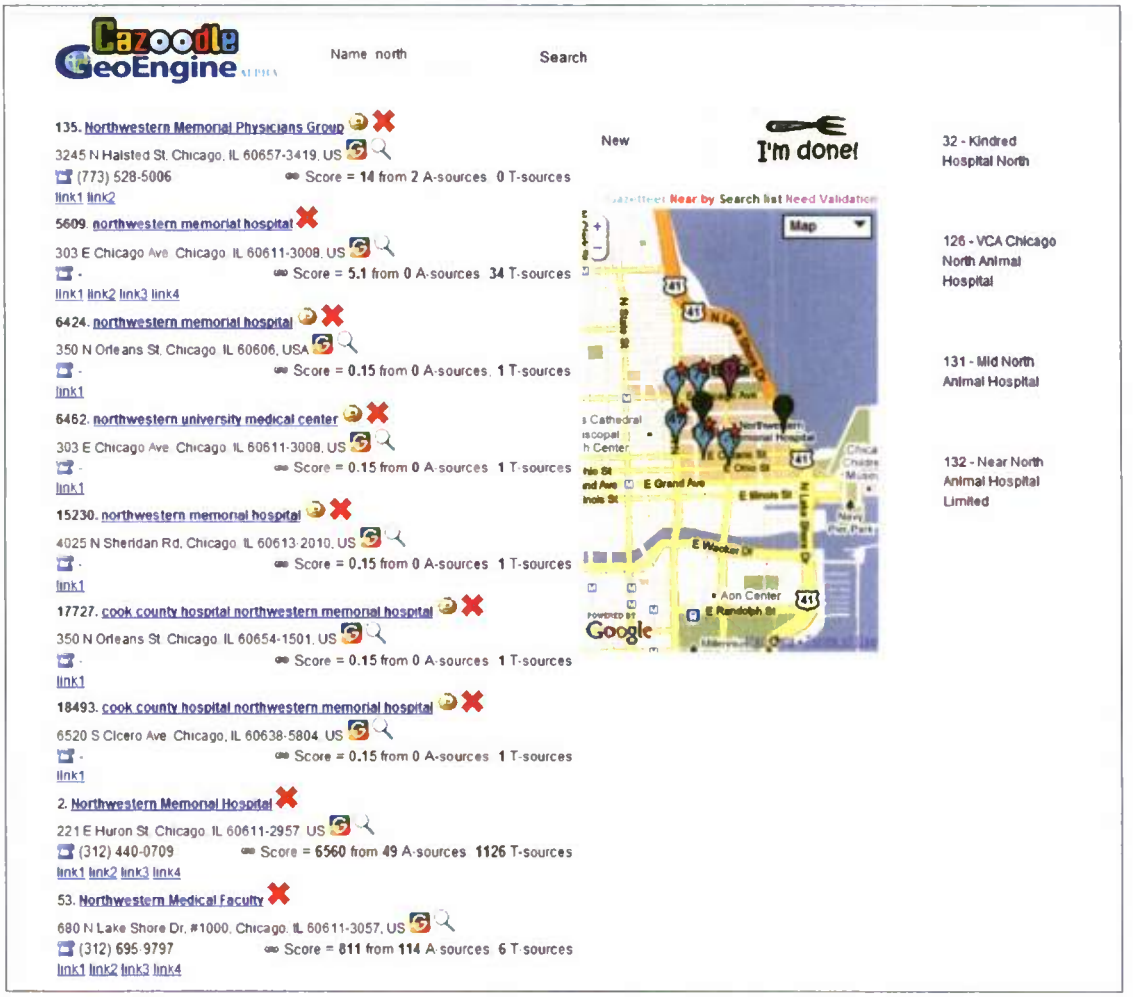


Figure 13: Browsing console for an analyst to use on the results produced by our algorithm.

- Each result includes an "X" button to delete that result. The analyst should determine that the result is really a hospital (*e.g.*, "the car hospital" is actually a car repair shop) and that the address of the result correctly represents the location of the hospital. If these conditions are not met, the result should be deleted.
- We highlight the results having low scores with a special "smiley" sign. This provides a cue to the analyst that this result has low algorithmic confidence, so there is a greater possibility that the result should be deleted.
- For each result we list hyper-links of the sources from which that hospital record was discovered by our algorithm. The analyst can click on these links to inspect the target sites and to determine if the hospital name is associated with the correct address.
- Each result includes a "G" sign, which will open a new browser window with results from a general Web search engine (currently set to *google.com*). We use the address of the hospital result as the keyword query. This helps the analyst find what facilities are located at a particular address, by looking at snippets of the search result or by inspecting a few of the topmost results. The analyst could further modify the query to add key terms from the name of the hospital result; the search engine would then likely return the pages that mention both the name and the address of that hospital result.

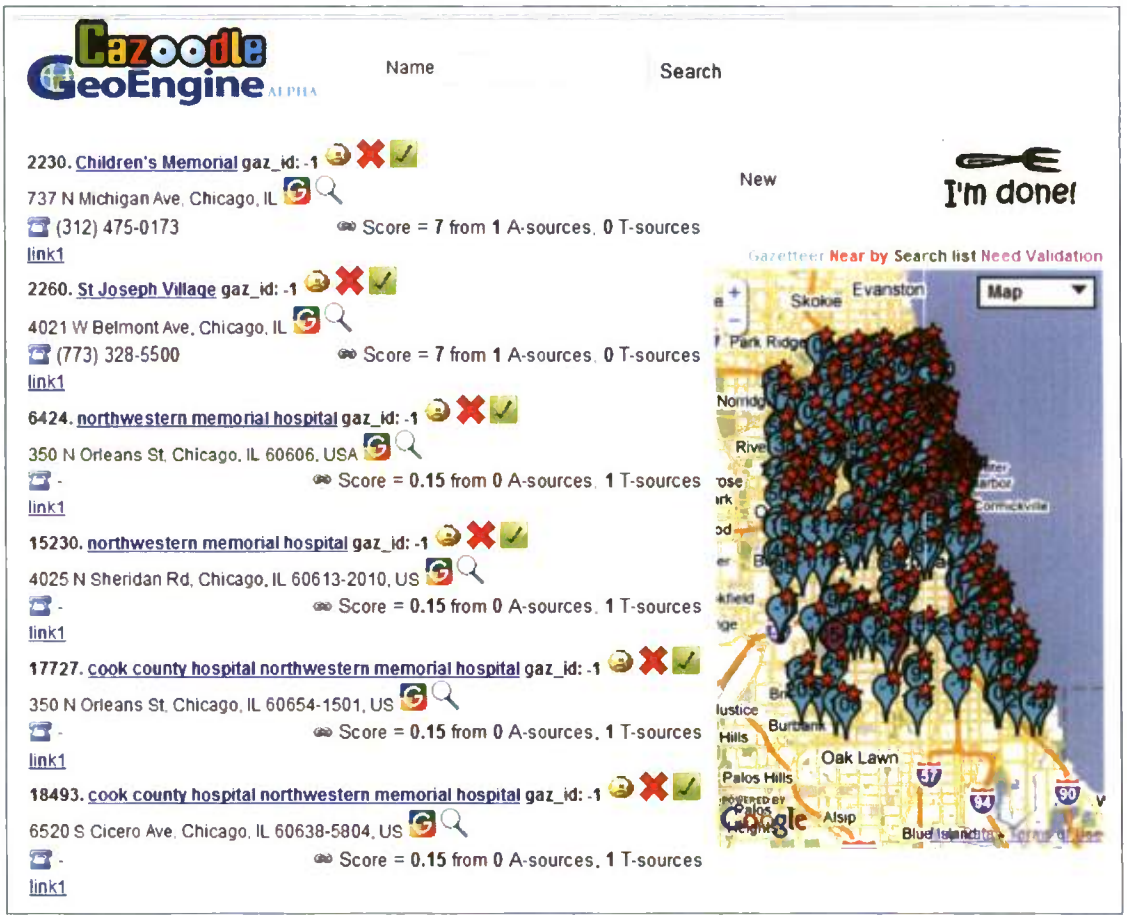


Figure 14: The Deletion Validation Console for a senior expert to confirm the deletion decisions of an analyst.

2.6.3 Merging Operation

After the deletion step, the analyst is left with results that are valid hospital features. Each of these results represents a group of raw records, merged by our GeoIntegration algorithm. When inserting these results into the final gazetteer, the analyst needs to verify if additional merging is needed. In particular, our algorithmic merging may produce more than one group for the same hospital feature. The analyst can further merge such results, as follows:

- Each result includes a "lens" icon, which zooms into the part of map surrounding the location of that result. It shows all the other results in the vicinity of the result, including those that are still in the search list, those that have been deleted, or those that have been inserted into the final gazetteer. This function helps an analyst determine whether to create a fresh entry for the result or to add it to an existing entry in the gazetteer.

- If the analyst determines that a hospital result represents the same feature as one of the entries that already exists in the gazetteer, the hospital result can be dragged and dropped into the box on the right panel, which corresponds to that gazetteer entry.

- If none of the existing entries in the gazetteer match the new hospital result, the analyst needs to create a new gazetteer entry for that result. The analyst can drag and drop the hospital result to the "New" box above the map in the center panel. This step will create a new entry for this result in the final gazetteer.

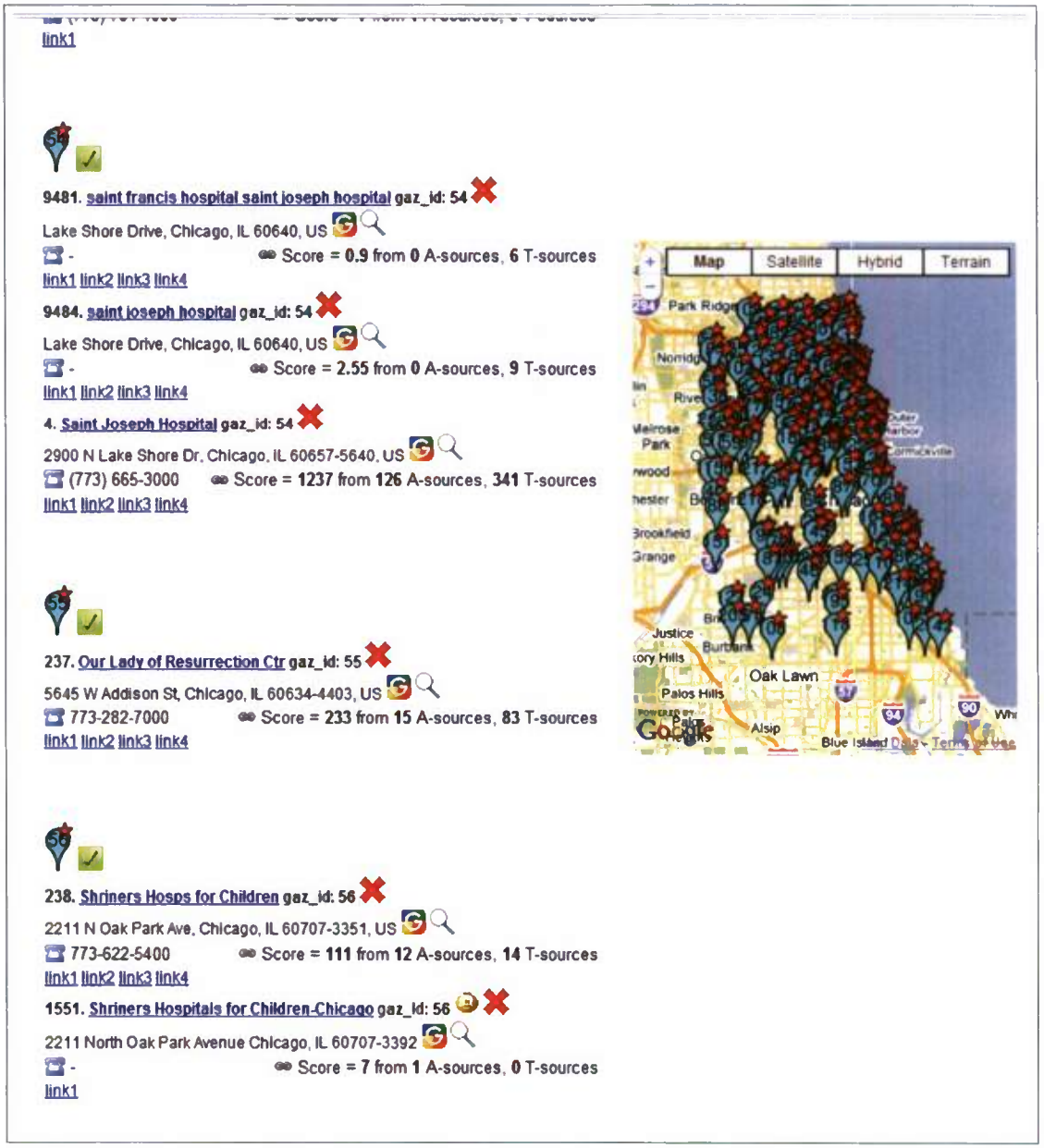


Figure 15: The Merging Validation Console for a senior expert to confirm the merging decisions of an analyst.

2.6.4 Validation Console

The Operations Console includes a final validation step for all decisions made by the analyst by means of the browsing console. We envision the overall process as being composed of two steps, in which first, a junior analyst generates an initial gazetteer, and then, the senior domain expert verifies these decisions. In situations in which the senior expert executes the first step with the browsing console, the validation step can be skipped and the initial gazetteer can be accepted as the final form.

- In the deletion validation console, as shown in Figure 14, we display all the hospital results that were deleted by the analyst from the browsing console. The senior expert verifies each deletion decision using procedures that are similar to those used with the browsing console. If the deletion is correct, the senior expert can

approve it. Otherwise the result can be sent back to the browsing console for the analyst to inspect again.

- In the merging validation console, as illustrated in Figure 15, the senior expert can see all the merging decisions made by the analyst. Each entry represents a gazetteer entry that is composed of multiple results produced by our algorithm. The senior expert can verify that the results grouped by the analyst represent the same feature. Also, the expert can verify whether the analyst failed to merge any results. Upon completion of this validation step, the final gazetteer will be produced and will be available for use by any intelligence application.

After operating on 351 results in the target geography, the analyst deleted 63 results, and merged duplicate results to produce 259 entries in the final gazetteer.

2.7 Deliverable Gazetteer

For the final gazetteer delivery, we used the Operations Console ourselves, "emulating" how an actual analyst would use it. The Operations Console was designed to be used in two steps: 1) For a junior analyst to construct a gazetteer using the hospital results generated by our automated techniques developed in the Phase I Option period, and 2) For a senior expert to verify the decisions made by the analyst. The resulting output was used as the final gazetteer.

We limited this study to a target area in a 5-mile rectangular bounding box around Chicago, as highlighted in Figure 11. Of the total of 4454 merged results produced by our algorithm for the Chicago metropolitan area, we found that 351 results fell into our target geography. The two steps of full deployment of the Operations Console for the final gazetteer delivery–for the junior analyst to construct gazetteer and the senior expert to validate decisions–each took one day of human effort.

An analyst inspected each of the 351 results, and a senior expert verified the decisions, resulting in the output of 259 hospital features. In comparison, the benchmark dataset from the US Geological Survey (USGS) contained only 94 features. Figure 12 shows a sample of entries from the final gazetteer. The full gazetteer is publicly available for download at `http://geoengine.cazoodle.com/data/finalgazetteer.txt`. Each final entry was obtained as a result of the merging of raw records from several online web sources. For example, the top entry, "Northwestern Memorial Hospital," was discovered from among 36 online sources. Overall, the 259 features were obtained from a total of 189 web sources.

2.7.1 Benchmark Performance Comparison

We used the hospital features in the USGS gazetteer as the benchmark for our evaluation and filtered the hospital features within our target geography. Upon filtering, we obtained 94 hospital features from the USGS gazetteer (available at `http://geoengine.cazoodle.com/data/usgsgazetteer.txt`).

Observation 1. Our final gazetteer covers 100% of the features in the USGS gazetteer.

We sampled 50 features from the USGS gazetteer and for each sampled feature we checked to see if our final output gazetteer contains that feature. We summarized our findings in Figure 16. Of the 50 sampled features, 2 hospitals have incorrect geo-coordinates in the USGS gazetteer. One of them, "St. Vincent Hospital," is actually in Massachusetts; another one, "Evangelical Hospital," is located in Pennsylvania. Of the remaining 48 features in the sample, our algorithm successfully discovered 42 features—finding an "exact match" for 34 features and a variant name for another 8 features. Our algorithm could not find the remainder of the 6 features; however, we observed that all of these 6 features represented hospitals that are no longer operational: some were shut down decades ago.

Overall, of the 50 hospital features that we sampled from the USGS gazetteer, only 42 were *valid* hospitals. The rest of the 8 features either had incorrect coordinates or were no longer operational. All of these valid hospitals were found in our output gazetteer; thus, the coverage of our algorithm is 100% with respect to the USGS gazetteer.

Observation 2. 16% of the features in the USGS gazetteer are inaccurate or out of date.

We note that one of the objectives of the USGS gazetteer is to explicitly mark the features that are not in operation anymore. For example, 6 of the 50 sampled features were explicitly (and correctly so) marked as historical, *e.g.*, "Martha Washington Hospital (historical)" or "Frank Cuneo Hospital (historical)." Our algorithm was successful in finding these features on the Web (among 42 matches).

Number of features in USGS	94
Number of features in our final gazetteer	259
Features sampled from USGS	50
Total valid features in sample	42
Feature with inaccurate coordinate	2
Feature with outdated information	6
Sampled features from USGS that are available in our gazetteer	42
Exact match available	34
A variant name available	8

Figure 16: Evaluation of our final gazetteer with respect to the USGS gazetteer.

Hospital from our gazetteer	Whether available in USGS gazetteer
Northwestern Memorial Hospital	√
Comer Children's Hospital	×
Cook County Hospital	√
Saints Mary and Elizabeth Medical Center	√
Saint Joseph Hospital	√
Illinois Masonic Hospital	×
Schwab Rehabilitation Hospital	×
Swedish Covenant Hospital	√
West Suburban Medical Center	√
Neurologic Orthopedic Hospital of Chicago	×
Rush University Medical Center	×
Loretto Hospital	×
Holy Cross Hospital	√
Louis A. Weiss Memorial Hospital	√
St Anthony Hospital	√
Norwegian American Hospital	√
Mercy Hospital & Medical Center	√
Mount Sinai Hospital Medical Center	√
Chicago Lakeshore Hospital	√
Methodist Hospital of Chicago	√

Figure 17: Evaluation of the coverage of the USGS gazetteer with respect to the top 20 hospitals in our final gazetteer.

We observed that the marking of the historical features is not up to date in the USGS gazetteer. Of the 8 features that our algorithm could not find, 6 represented the hospitals that are no longer in operation (some were shutdown decades ago). The USGS gazetteer, however, failed to mark them as "historical."

We also found that the coordinates of some of the features in the USGS gazetteer are incorrect. Of the 8 features in the sample set that our algorithm could not find, 2 of them have incorrect coordinates in the USGS gazetteer. One of these hospitals is located in the state of Pennsylvania, while the other is located in the state of Massachusetts. In the USGS gazetteer, the coordinates of these two features are incorrectly set as located to be in the vicinity of Chicago (and that is why they fall within our target geography).

To summarize, the USGS gazetteer has inaccurate or outdated information for 8 out of the 50 sampled features, *i.e.*, for 16% of the cases. We believe our algorithm could provide great assistance in automatically identifying entries in the USGS gazetteer that may need correction. If our algorithm could not find a feature from the Web, that feature in the USGS gazetteer is possibly either out of date or has incorrect coordinates.

Observation 3. The USGS gazetteer covers only 70% of the top 20 features in our final gazetteer.

In another study, we wondered how extensive the coverage of the features in the USGS gazetteer might be. We took the top 20 features from our final output gazetteer, ranked in the order of the number of sources in which we found a feature. As shown in Figure 17, the USGS gazetteer does not contain 6 of these 20 features. This

Source Name	Number of features in final gazetteer
local.yahoo.com	245
maps.google.com	46
www.revolutionhealth.com	41
allhospitals.org	41
ushospitalfinder.com	39
www.hospitalsworldwide.com	39
www.doctordirectory.com	38
www.mchc.com	38
www.dogster.com	33
www.idph.state.il.us	31
www.hospitalcompare.hhs.gov	30
www.vimo.com	25
www.healthcarehiring.com	24
www.cazoodle.com	22
hospitalandmedicalcentercompare.com	21
health.usnews.com	17
www.hospitalsoup.com	10
www.yelp.com	9
alcoholism.about.com	9
32 sources	[2, 5] features
137 sources	Only 1 feature

Figure 18: Distribution of the number of features extracted from each contributing source.

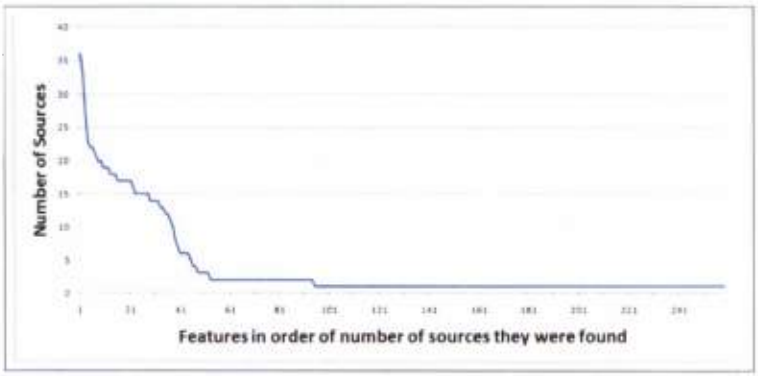


Figure 19: Distribution of the number of sources from which each of the features were extracted.

indicates that the coverage of the USGS gazetteer is quite poor. We believe that our automatic techniques are necessary to ensure the completeness of databases such as the USGS gazetteer.

2.7.2 Distribution Statistics

Besides the benchmark evaluation, we also analyzed our final gazetteer to identify the merits of our approach.

Observation 1. Aggregation of content across many sources is necessary.

Our final gazetteer was composed of 259 features that were discovered across 189 Web sources. We analyzed the contribution of each source, *i.e.*, for each source we studied a fraction of the 259 features that were found at that source. As Figure 18 shows, the most popular source was *yahoo.com*, where we were able to find 245 out of the 259 features. With the exception of this source, the remainder of the sources covered far fewer features. The second most popular source was *google.com*, where we were able to find 46 of the 259 features. In fact, a large

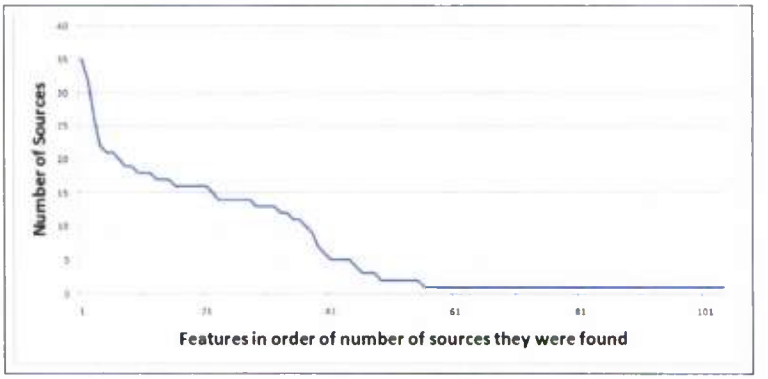


Figure 20: Distribution of the number of sources from which each of the features were extracted, after excluding *yahoo.com*.

Number of algorithmic results merged manually	Number of such entries in final gazetteer
5	2
3	3
4	3
2	6
1	245

Figure 21: Distribution of the number of algorithmic results that were merged manually per entry in the final gazetteer.

majority of the sources, *i.e.*, 137 out of the 189 sources, contributed only one hospital feature apiece. Only 19 sources contributed more than 5 hospital features.

In another study we analyzed the number of sources in which each of the 259 features were found. As shown in the distribution in Figure 19, we found that the most popular feature could be found in only 36 out of the 189 contributing sources. A large majority of features, *i.e.*, 165 out of 259 features, were found on only one of the contributing sources. To avoid confusion, we should note that the specific source in which these 165 features were found may be different for each of these features.

To remove the skew due to one exceptionally large source (*yahoo.com*), we also analyzed the distribution after excluding *yahoo.com* from our study, *i.e.*, considering only those features that could be found from at least one of the remaining 188 sources. Of the 259 features, 155 features could be found only on *yahoo.com*. After excluding these features from our study, we looked at the distribution for the remainder of the 104 features that could be found in at least one other source besides *yahoo.com*. As shown in Figure 20, the most popular feature could be found from only 35 sources. A vast majority of the features– as many as 49 out of the 104 features–could be found only in one source, which implies that the only way to generate a complete gazetteer is by aggregating information discovered across all sources.

Observation 2. The automatic merging technique is effective.

Next, we studied the effectiveness of our automatic merging algorithm by evaluating how much manual effort was required by the analyst for generating the final output gazetteer. Starting from the results produced by our automatic techniques, our analysts performed two key manual operations: deletion of the incorrect results and manually merging results that our algorithm could not merge automatically.

We found that our algorithms for automatic merging were quite effective. Figure 21 shows the distribution of the number of results produced by our algorithm that were grouped together by the analyst to produce an entry in the final gazetteer. For instance, the first row shows that there were 2 entries in the final gazetteer for which our analyst had to manually merge 5 results each. For a large fraction of the gazetteer entries, *i.e.*, 245 out of the 259 features in our final gazetteer, the analyst did not need to perform any manual merging. This indicates that our algorithm is quite effective in automatically determining which raw records need to be merged.

Observation 3. Manually deleted entries can be automatically filtered by new improvements in the

Reason for error	Number of results
Address of a hotel	23
Other businesses (drugstore, associations, car repair, computer repair)	16
Incomplete extraction	12
Incorrect association	6
Senior or disability housing	6
Total results deleted by analyst	63

Figure 22: Analysis of the reasons for erroneous entries deleted by analyst.

text extraction module.

Next, we analyzed how much difficulty our analyst faced in deleting erroneous results produced by our algorithm. Of the 351 results produced by our algorithm in the target geography (defined by the bounding box), our analyst deleted 63 results. The remainder of the results were further merged, finally producing 259 entries in the output gazetteer. Thus, only about 18% of the results produced by our algorithm were erroneous and therefore needed to be deleted by our analyst.

We further analyzed the characteristics of these 63 results deleted by our analyst, as summarized in Figure 22.

- We found that 23 of the deleted results represented cases in which a hotel was located at the address of the candidate feature. We found that many online websites that provide search functions to find hotels displayed lists of hospitals near those hotels. When we applied our text extraction module to these pages, we incorrectly associated the address of the hotel with the hospital name on that page. We can enrich our text extraction technique by segmenting a Web page into units of coherent information and disallowing associations across different units.

- We found that 16 of the deleted results either represented real-world services whose names were ambiguous, *e.g.*, "Car Hospital," or "The Computer Health Center," or represented services related to the healthcare industry, *e.g.*, drugstores, educational institutions, or recruiting agencies for the hospital industry. Our analyst could very quickly judge that such results were erroneous, thus requiring little overhead. We can also develop techniques to filter out such results automatically–by first training a language model for the Web pages representing hospital features, and next, comparing the context of the new Web pages to the trained model.

- We found that 18 of the deleted results represented cases in which our extraction techniques could be improved, *i.e.*, either the name of a hospital was incorrectly associated with the address of a different hospital (6 cases), or the extracted address was incomplete (12 cases). In our implementation of the text extraction module for the hospital task, we generated all pairs of hospital names and addresses as candidate records, *i.e.*, a "full-join" of all hospital names and all addresses in a single page as candidate features. We used the number of *web-references* by using the number of results returned by search engines for each pair to compute the score for each. While the scoring scheme already performed reasonably well, we can improve accuracy by pruning the associations that do not qualify as the tightest pair, *e.g.*, if the text segment between a candidate pair of a hospital name and an address included another hospital name (or address), then the pair would not satisfy the tightest binding requirement.

- We found that 6 of the deleted results represented residential housing facilities for senior citizens or people with disabilities. Often, some of these services also include an adjoining medical facility. Such results are perhaps best left for the analyst to judge with regards to whether they should be included in the final gazetteer.

3 Mosque Feature Discovery and Extraction

In the first part of the Phase II work, we attempted to solve concrete feature discovery problems of various types: *man made* features (*e.g.*, hospitals, places of worship), *natural* features (*e.g.*, mountains or lakes), and *colloquial* features (*e.g.*, neighborhood boundaries). We focused on two countries: Afghanistan and United States.

	English		Arabic	
1	Afghanistan mosque	2,260,000	افغانستان مسجد	1,460,000
2	mosque finder	524,000	مسجد مكتشف	491,000
3	mosques finder	216,000	مكتشف المساجد	132,000
4	mosques finder Afghanistan	194,000	المساجد مكتشف افغانستان	65,600
5	masjid finder Afghanistan	15,200	مسجد مكتشف افغانستان	59,900
6	holy place Afghanistan	67,400	"المكان المقدس" افغانستان	533
7	place of worship Afghanistan	290,000	مكان للعبادة افغانستان	79,800
8	dargah Afghanistan	118,000	دار غا أفغانستان	96
9	house of worship Afghanistan	211,000	بيت للعبادة افغانستان	303,000
10	Network of Mosques (Afghanistan)	360,000	شبكة من المساجد	896,000

Figure 23: Comparison of number of results on *google.com* for English vs. Arabic queries.

We first studied a series of concrete feature discovery problems, for man-made, natural, and colloquial features – thus, discovering mosque features is our first concrete problem, in the *man-made* category of features, after our development of the hospital discovery system in the interim Phase I Option period.

This problem of mosque discovery is quite different from hospital discovery, in two aspects:

1. Language difference. While the hospital discovery system operated on an English language corpus, the mosque discovery system may need to access foreign language (*i.e.*, Arabic) Web pages.

2. Geography difference. Our hospital discovery task focused in the Chicago metropolitan area, where effective geo-coding tools are available. How well can these tools adapt to the new target geography of Afghanistan?

Additionally, we worked on the problems of information extraction, feature merging, evaluation, user-aided analysis and made additional improvements to our aggregation engine.

3.1 Language Difference

3.1.1 Findability Survey for Mosque Information

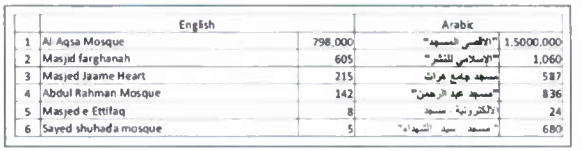
We studied the language nature of Web pages that will be useful for the discovery of mosque features. For our survey, we queried *google.com* with different keywords relevant to our domain of interest. We summarize the findings as follows:

Observation 1: Mosque information is generally prevalent in both English as well as Arabic pages. We searched for several "mosque-related" keyword queries, in English as well as their corresponding Arabic translations, on *google.com*. In Figure 23, we show the number of results returned for each of these queries. We see that both English and Arabic queries match a large number of pages. For some queries, we saw a higher result count for English queries, and vice versa for the others. This phenomenon indicates that mosque information is widely available in both English and Arabic language corpus.

Observation 2: The information for a specific mosque may be more prevalent in Arabic than in English pages. We queried *google.com* with a few specific mosque names as queries. As Figure 24 shows, we found more results in the Arabic versions than the English versions. This phenomenon indicates that we will likely find more information in Arabic pages for a *particular* mosque. Thus, even for mosques (and other associated information) that we could discover in English-language pages, we will find more information in Arabic pages for further augmentation.

Observation 3: Mosque information is available in both structured Web sites and unstructured text pages. We inspected a few results for each of our different survey queries. From a total of about 1000 sources thus inspected, we report the characteristics of a few relevant sources in Figure 25. As we observe, there is a good mix of structured sources and text pages—thus, it indicates the need for the dual-extractor design for this problem as well.

Observation 4: The exact information about the locations of mosques is rarely available. As Figure 25 shows, the full addresses of mosques are often not available. Some sources only provide city-level addresses, and some provide "relative" addresses (*e.g.*, 50 km northeast of *Sakhu*).



	English		Arabic	
1	Al Aqsa Mosque	798,000	"الأقصى المسجد"	1,5000,000
2	Masjid farghanah	605	"الإسلامي للنشر"	1,060
3	Masjed Jaame Heart	215	مسجد جامع هرات	587
4	Abdul Rahman Mosque	142	"مسجد عبد الرحمن"	836
5	Masjed e Ettifaq	8	الكترونية مسجد	24
6	Sayed shuhada mosque	5	"مسجد سيد الشهداء"	680

Figure 24: Comparison of number of results on *google.com* for English and Arabic versions of specific mosque names.

Site	Format	Mosque Names	Address
islamicfinder.org	structured	> 10	full address
wikipedia.org	text	> 10	city only
gearthhacks.com	structured	$1 - 10$	city only
lib.uwm.edu	structured	$1 - 10$	city only
mfa.gov.af	text	$1 - 10$	city only
archnet.org	structured	$1 - 10$	city only
orientalarchitecture.com	text	$1 - 10$	no address
ramdan4u.blogspot.com	text	$1 - 10$	city only
allexperts.com	text	$1 - 10$	city only
aulia-e-hind.com	text	$1 - 10$	city only
5ymah.net	text	$1 - 10$	city only
trytop.com	text	$1 - 10$	no address
afghan-network.net	text	$1 - 10$	relative address

Figure 25: Nature of sources providing mosque information.

3.1.2 Text Extractor for Arabic Pages

As our survey above shows, mosque information is quite prevalent in Arabic pages. We thus decided to investigate how well our extraction techniques can extend to a foreign language like Arabic and to understand what the issues are. Therefore, we attempted to customize our text extraction techniques to recognize mosque names from Arabic pages, which were collected from the survey of Arabic keyword queries as just reported. This study would not be possible without the knowledge of Arabic—we have a colleague who is fluent in the Arabic language, and he helped us in this exercise. Our lessons from this exercise provided a mixed experience.

Lesson 1. The state-of-the-art language translation tools are not effective for the purpose of information extraction across languages. Initially, we thought we could simply translate the Arabic pages to English, and then use our text extraction tools on the translated pages. We tested the translation effectiveness of *Google Translate*. First, we found the response time of the service to be too slow for large scale processing. It took 10-15 seconds to translate a typical Web page. Secondly, we found the accuracy of translation to be quite limited—it could only translate some parts of the pages. Our Arabic expert inspected the results and found that the original Arabic pages often used a variety of local dialects—thus, the translation problem is inherently difficult. As there is active research going on in natural language processing for the Arabic language (*e.g.*, [1]), we are hopeful that better tools would soon become available in the future, although currently this translation-then-extraction approach does not seem viable.

Lesson 2. The parsing module can be easily extended to operate on Arabic text. Although the Arabic language is quite different in its composition – it is written from right to left, and uses Unicode encoding – our parsing module could be extended to tokenize Arabic Web pages. We were able to recognize the Arabic word "mosque" in Arabic pages, and segment the surrounding context of the matching word tokens. Our Arabic expert observed these segments of the extracted tokens, and found the matches to be fairly accurate.

Lesson 3. The adaptation of extraction rules to Arabic requires much knowledge of the target language. As the step after segmenting the surrounding context of the Arabic "mosque" keyword, for each such segment, we needed

language-specific rules for actually recognizing mosque names. For instance, a rule can state if there is "Al-" (the definite article in the Arabic language) at two or three tokens preceding the "mosque" keyword, then the character sequence in between them would be the name of a mosque.

As we realized in our exercise, constructing such rules requires good knowledge of the grammar and conventions of the target language. Since the issues here are rather language specific, we would leave such customization for future extension. Meanwhile, better tools for Arabic information extraction will become available, since name entity recognition for the Arabic language (and various other languages) is actively pursued in research (*e.g.*, [45, 46, 72, 88]). Thus, lacking foreign language expertise, we decided that, for our study in the *GeoEngine* project, we will focus on developing extraction modules for English corpora.

Lesson 4. The ability to parse Arabic pages can be used for augmenting English-based feature discovery. While we will not study the extraction of information directly in Arabic pages, we can use these pages to augment the features discovered from English pages. As we saw in Lesson 2, we could successfully parse and identify desired tokens in Arabic pages. We can thus use this ability to find pages containing auxiliary information for mosque names that we discover from an English corpus. These matching pages can be used to augment in the discovery process.

3.1.3 Language prevalence

A major impact of change in geography from Chicago to Afghanistan is the change in language–from English to Arabic. We queried *google.com* to estimate the number of pages, in Arabic vs. in English, that provide mosque related information. To recap, we queried *google.com* with the English as well as the Arabic variants of several mosque related queries. Figure 23 summarizes the number of results found for 11 general queries, while Figure 24 shows the results for 6 queries with specific mosque names.

Observation 1: There is a good mix of English as well as Arabic language pages providing mosque information. In general, we observe that the mosque information is prevalent in both the languages. For the general queries, English corpus is more popular, while for the specific mosque names, Arabic corpus is more popular.

3.1.4 Nature of sources

To understand the nature of the Arabic sources that are returned in search results for our queries, we inspected the top 10 results for each of them (or all results for queries where less than 10 results were found). The results are summarized in Figure 26. We used *Google Translate* `http://translate.google.com` to translate these pages from Arabic to English, for our inspection. If a page could not be translated, we mark it as "GT Failed." Strangely, we also observed quite a few links were now defunct, marked as "Link Down." A source is classified as "DB" if it is a structured site, suited for our agent extraction. The rest of the sources are text sources, further classified as {wiki, travel, news, blogs, org, forum, culture}, depending on their characteristics.

As Figure 26 shows, of the top 10 results for $Q1$, 1 is Link Down, 1 is GT Failed, and rest of the 8 results are text sources, with no DB sources. The first 11 rows are for general queries, and the next 6 rows are for the queries with specific mosque names.

Observation 2: We rarely find structured DB sources in Arabic language. For the general queries, we never saw any DB sources. For the specific queries, we saw DB sources for 4 out of 6 cases. These 4 results belonged to Arabic version of 2 structured sources – *wikipedia* and *islamicfinder*. In addition to this survey, we inspected many more results for discovery of structured sources. Yet, we could find a total of only 4 structured sites.

Observation 3: Casual channels are predominant source of information. Of the 165 sources that we inspected, 50 sources were forum discussions. The second and third most popular information source are blogs and news sites. Together, forums, blogs and news sites comprise of 104 out of 165 sources.

Observation 4: Language translation does not work on all pages. *Google Translate* was not successful for 14 of the 165 sources that we inspected.

Observation 5: Some sites are not well maintained. Even though our survey was restricted to the top 10 results, we still saw 13 of the links did not load – either throwing HTTP 404 error, or internal server error.

Thus, the text sources were the major focus for development of *GeoEngine* for Mosques in Afghanistan.

Query	wiki	travel	news	blog	org	forum	DB	culture	GT Failed	Link Down
General Queries										
Q1		1	3	1		3			1	1
Q2			1	3		6				
Q3	1	1	1	1		4			1	1
Q4				5		4				1
Q5		1	1	2		2		1	1	2
Q6			2	1		3			2	2
Q7		1	2	3		2			1	1
Q8			2	1		4		1	1	1
Q9			1	1		1			2	
Q10		1	4			3			1	1
Q11				3	1	4			1	1
Specific Mosque Queries										
Q12	3	1		1			2	1	2	
Q13	2	1	1	1		2	1	1		1
Q14	2			2		5	1			
Q15	2	1	1	1		2	1	1	1	
Q16			3			4		2		1
Q17	1	2	4	2		1				
Total 165 pages	11	10	26	28	1	50	5	7	14	13

Figure 26: Study of the nature of top 10 results for mosque related queries.

3.1.5 Language translation

Observation 6: The languages spoken in Afghanistan are not directly supported by state-of-the-art translation tools. While the language translation tools support Arabic to English translation, Afghanistan uses other languages, which limits the accuracy of translation services. The two most popular dialects are Dari (also called Farsi) and Pashto (also called Persian), used 50% and 35% respectively. While both of them use Arabic alphabet, they differ considerably from original Arabic language. The other popular languages are Turkic languages (Uzbek and Turkmen) and several minor ones, *e.g.*, Baluchi, Pashai, Nuristani, *etc.*.

Observation 7: The approach of translate-then-extract is not viable for Arabic corpus. We found two good resources for Arabic to English translation– *Systran* and *Google Translate*, and the later seems to be the better. However, due to variations in local dialects (as noted in Observation 6), sometimes large parts of the text could not be translated (as seen in Observation 4). Due to these limitations, the translate-then-extract approach does not seem viable currently.

We also conducted a "fun" experiment to use *Google Translate* to translate names of a few mosques from English to Arabic, and then back to English. The results, as summarized in Figure 27, show some funny translations, *e.g.*, "Friday Mosque of Herat" becomes "A mosque on Friday in Herat," or, "Pul-e-Khishti Mosque" becomes "E, in a mosque Khishti." These examples also illustrate the inherent limitation of the state-of-the-art tools.

Observation 8: Adaptation of text extraction modules for Arabic corpus requires much language

Original English name	Translated Arabic Name	Reversely translate to English
Puli Kheshtee Mosque	مسجد Kheshtee بولي	Poly Kheshtee mosque
Masjid i Khwaja Abu-Nasr-i-Parsa	مسجد أبو النصر ، ط بارسا ط خواجا	I Khwaja Abu al-Nasr mosque, I Parsa
Masjide farghanah	Masjide farghanah	Masjide farghanah
AL-JIHAD MOSQUE	مسجد الجهاد	Jihad Mosque
CENTRAL JAMIE MOSQUE	مسجد جيمي سنترال	Jimmy Central Mosque
Masjed Jameh shahri	الشهري جامع مسجد	Mosque Mosque monthly
Masjid Jamee Taluqan	تالوكان جامع مسجد	Mosque Mosque Taloqan
Abdul Rahman Mosque	الرحمن عبد مسجد	Abdel Rahman mosque
Friday Mosque of Herat	هيرات في الجمعة يوم مسجد	A mosque on Friday in Herat
Green Mosque	الخضراء مسجد	Green Mosque
Haji Piyada	حاجي Piyada	Haji Piyada
Khost Mosque	مسجد خوست	Khost Mosque
Lashkar Gah Mosque	مسجد لشكركاه	Lashkar Gah Mosque
Mosque of the Hair of the Prophet	للنبي الشعر من مسجد	Mosque of the hair of the Prophet
Pul-e Khishti Mosque	مسجد في ، ، Khishti	E, in a mosque Khishti
Shrine of Hazrat Ali	علي حضرة ضريح	The shrine of Hazrat Ali
Id Gah Mosque	مسجد جاه معرف	ID Gah Mosque
Khwaja 'Abd Allah Ansari shrine	مزار الأنصاري الله عبد خواجا	Khwaja Abdullah Ansari Shrine

Figure 27: Results of *Google Translate* from mosque names in English to Arabic to English.

insight. Having observed the difficulty in language translation, we explored the direction of extending our text extraction modules directly into Arabic corpus. Our experience was mixed–while it is technically possible to extend our tools to work on foreign language corpus, the adaptation requires much insight into Arabic language alphabet, syllable, vocabulary, and grammar.

Thus, for our development, we will focus only on English corpus pages, leaving Arabic corpus to be incorporated in future, as better translation tools become available.

3.2 Information Extraction

3.2.1 Information availability: Chicago vs. Afghanistan

With our focus on English corpus, next we wanted to zoom deeper to understand what type of information is available on the Web for mosques in Chicago vs. mosques in Afghanistan.

For our study, we used 5 popular mosques in Chicago and Afghanistan, each, and designed two experiments. As our first experiment, we queried *google.com* for the specific mosque names and noted the number of matching results, as shown in Figure 28. These numbers are indicative of number of web pages providing information relevant to that mosque.

Observation 9. The number of web pages providing mosque information is far greater for mosques

Geography	Mosque Name	Result Count
Chicago	Mohammad Islamic Corporation	65
	As-Salaam Center	109
	Roscoe Mosque	293
	Alsalm Mosque Foundation	128
	Makki Masjid Incorporated	65
Afghanistan	Ibrahim Shah Baba Shrine	6
	Dara Herat Mosque	2
	Khost Mosque	590
	Friday Mosque	64,500
	Lashkar Gah Mosque	234

Figure 28: Information availability: Contrasting number of results found on *google.com* for 5 mosques in Chicago and Afghanistan, each.

Name	city	Address	Prayer time	Picture	Phone
As-Salaam Center	Yes	Yes	Yes	Yes	Yes
Roscoe Mosque	Yes	Yes	No	Yes	No
Alsalm Mosque Foundation	Yes	Yes	No	No	Yes
Makki Masjid Incorporated	Yes	Yes	No	No	Yes
Muhammad Islamic Corporation	Yes	Yes	No	No	Yes

Figure 29: Availability of information for mosques in Chicago.

in Chicago compared to mosques in Afghanistan. As summarized in Figure 28, we see far more results for mosques in Chicago compared to mosques in Afghanistan. The only exception is "Friday Mosque," for which we get high number of results at 64,500; however, this is a quite common mosque name, with many cities around the world having a mosque with this name (see the *wikipedia* article [2]for more info).

As our second experiment for the information availability, we tried to understand what type of information can we find for mosques on the Web. In particular, for the same 5 mosques in Chicago and in Afghanistan, our human experts manually tried to find–(#city, #address, #prayer-times, #pictures, #phone). The results for Chicago and Afghanistan are shown in Figure 29 and Figure 30, respectively.

Observation 10. The phone number and street addresses are rarely available for mosques in Afghanistan, while commonly available for mosques in Chicago. As summarized in Figure 31, phone number could not be found for any of the 5 mosques in Afghanistan, while could be found for 4 out of 5 mosques in Chicago. Also, the street address could be found for only 1 out of 5 mosques in Afghanistan, while it was easily discovered for all 5 mosques in Chicago.

This observation implies new challenges for our Mosque in Afghanistan system–On the one hand, we need to simplify our goal to discover only #city level coordinates for a mosque. On the other hand, we need to design new geo-integration technique since the integration technique used in Mosques in Chicago system relies on the street address, phone number and exact geo-coordinates.

Observation 11. Surprisingly, the pictures are more commonly available for mosques in Afghanistan as compared to mosques in Chicago. As we see from Figure 31, pictures could be found for 4 out of 5 mosques in Afghanistan, while for only 2 out of 5 mosques in Chicago. This indicates photo sharing sites could be useful for our development.

[2] *http://en.wikipedia.org/wiki/Friday_Mosque*

Name	City	Address	Prayer time	Pictures	Phone
Ibrahim Shah Baba Shrine	Yes	No	No	Yes	No
Dara Herat Mosque	Yes	Yes	No	No	No
Khost Mosque	Yes	No	No	Yes	No
Friday Mosque	Yes	No	No	Yes	No
Lashkar Gah Mosque	Yes	No	No	Yes	No

Figure 30: Availability of information for mosques in Afghanistan.

	city	Address	Prayer time	Picture	Phone
Chicago	5/5	5/5	1/5	2/5	4/5
Afghanistan	5/5	1/5	0/5	4/5	0/5

Figure 31: Contrast of availability of information for mosques in Chicago vs. in Afghanistan.

3.2.2 Geocoding accuracy

As our final dimension of survey for change in geography, we studied how well the geocoding services that we use (*Google Maps* and *Yahoo Maps*) work for Afghanistan, our new target geography.

Observation 12. The geocoding services are not capable of geocoding street addresses in Afghanistan. Although, it is hard to find street addresses for mosques in Afghanistan (as noted in observation 10), we could find the addresses for 3 mosques, as listed in Figure 32. We could not successfully geocode these these street addresses using either *Google* or *Yahoo*. *Google* returned empty results for all the 3 test cases; *Yahoo* returned city level geocodes for the 3 test cases.

Observation 13. The geocoding services are not capable of geocoding even the city names in Afghanistan. We used NGA for AF as reference DB, and evaluated how well the geocoding service from *Google* can support the cities in NGA. The findings are summarized in Figure 33. We sampled 200 features from NGA. For each feature, we queried the geocoding service with pattern "{city-name}, Afghanistan." We found *Google* could geocode only 59 of the 200 city names, returning empty results for rest of the 70% queries. Furthermore, even for these 59 cases where geocoding was successful, the distance between the coordinates returned by *Google* vs. NGA were more than 10 miles away for 41 cases. In fact, for 2 cases, the geocodes returned were outside Afghanistan (Pakistan for both).

3.3 Text Extraction

The design of text extractor, as illustrated in Figure 34 on a page from *wikipedia*, involves three steps: (a) City Name Annotation to locate mentions of the Afghanistan cities on that page, *e.g.*, "Kabul Bazaar" (C1), "Kandahar" (C2), *etc.*, (b) Mosque Name Extraction to recognize the names of the mosques, *e.g.*, "Mosque of the Hair of the Prophet" (M1), or "Jame Mui Mobarak" (M2), *etc.* and (c) Mosque Tuple Assembly to combine the instances of mosque names (M1, M2, *etc.*) and city names (C1, C2, *etc.*) to produce mosque tuples of the form (#mosque-name, #city).

Address	Google Place	Google Lat	Google Lng	Yahoo Place	Yahoo Lat	Yahoo Lng
Jadde lilamiha, Herat, Herat 009340, AFGHANISTAN		0.00	0.00	Herat, Afghanistan, AF	34.35	62.19
Masjid i Khwaja Abu-Nasr-i-Parsa the center of Balkh City, Balkh, AFGHANISTAN		0.00	0.00	Balkh, Afghanistan, AF	36.75	66.90
District 7 Kabul Chahar Dihi, Kabul, Kabul , AFGHANISTAN		0.00	0.00	Kabul, Afghanistan, AF	34.53	69.14

Figure 32: Performance of Geocoding services in supporting street addresses in Afghanistan.

Features sampled from NGA	200
Geocoding returned empty result	141 (70.5%)
Certainly incorrect geocode (distance between geocodes and NGA $>$ 10 miles	41 (20.5%)
Possibly correct geocode (distance $<$ 10 miles)	18 (9%)

Figure 33: Performance of Google geocoding service in geocoding Afghanistan cities.

3.3.1 City Name Annotation

We used NGA Gazetteer of Afghanistan to obtain the list of all cities in Afghanistan. As learned from our survey, full street addresses are rarely available; so our module of regular expression based address recognition, that we used in our *Hospital in Chicago* system, is not applicable here. Using the list from NGA Gazetteer, we used GATE annotator to mark-up all the instances; and used Lucene to create inverted-index for easy access to the annotated entries.

3.3.2 Mosque Name Extraction

Our module for Hospital Name Extraction could be easily adapted to this new task. The method is based on defining a "state machine" as follows:

1. Look for key terms. We parsed the HTML string of the web pages to obtain clean text string. In this text string, we looked for terms in a manually compiled list of *MosqueTerms* = {"mosque," "masjid," "shrine"}.

2. Prune tokens beyond name boundary. We considered the tokens in the surrounding context of the key terms (up to 5 token positions) as potential mosque name. Within this sequence of tokens, going away from the occurrence of key term, we look for special tokens that indicate the boundary of mosque names—*stop words* (*e.g.*, of, an, the, etc.), or *punctuations* (*e.g.*, ., ! etc.), or *digits*, or *verbs*. Currently, we populate the the list of "name boundary" tokens manually; we hope to incorporate natural language annotation tools to help in simplifying this process.

3. Accept under special contexts. We identify some cases that are exceptions to the "name boundary" step, by checking the context around the name boundary tokens. That is, we recognize that some boundary tokens may not indicate name boundary under special situations of the surrounding context. For example, if the stop word is "of" and the token before it is another key term (*e.g.*, "mosque"), then we do not treat "of" as the name boundary (*e.g.*, in case of, "the mosque of the hair of prophet").

3.3.3 Mosque Tuples Assembly

Having extracted the list of mosque names (*e.g.*, M1, M2, ...) and the list of city names (*e.g.*, C1, C2, ...), our next step is to determine which of the "candidate tuples" (obtained as cross-join of the two lists) represent meaningful association.

1. Proximity-based Scoring

   While the potential number of candidate tuples are huge, to determine which of the candidate associations are meaningful, we use the *EntityRank* technique, which we developed in prior research in the WISDM

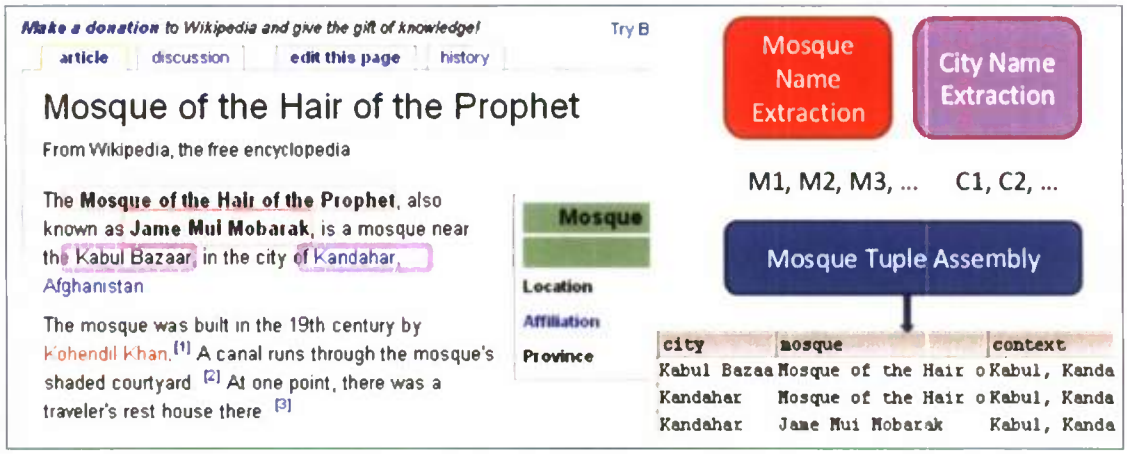


Figure 34: The design overview of Text Extractor for unstructured pages.

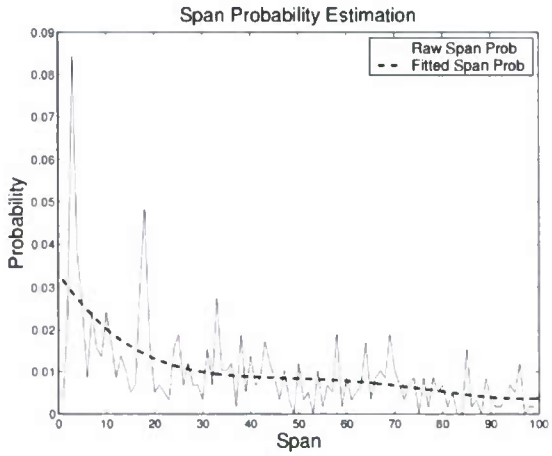


Figure 35: The span proximity model: Associating probability vs. span distance.

project ([55, 56, 57]) at the University of Illinois, and also used in our Phase I task of finding population. As a generic mechanism, EntityRank essentially explores the *proximity* of participating entity instances to score their association into tuples. To quantify proximity, we measure the *span* of the entity instances' occurrence positions, *i.e.*, the length of a window (in terms of the number of words) that covers all of them. While this proximity is an intuitive heuristic, to illustrate, we empirically measure the probability of manually-labeled associations versus span distances, for real world tuples like companies and their service phone numbers, *e.g.*, (IBM, 877-426-2223), over a real Web corpus. As Figure 35 displays, the association probability diminishes as the distance increases, which is nearly zero when the distance is greater than 100 words apart. We thus quantify the score of a potential tuple by the span of terms (the closer, the better) and the exact scoring can be empirically determined, much like what Figure 35 sketches.

2. Geo-Specialized Scoring.

   (a) Tightest-tuple Binding.

   Besides the generic EntityRank, we also enforced tightest binding of entities. More specifically, a candidate tuple (#mosque-name, #city) is valid only if there are no occurrences of other #mosque-name or #city instances between these tokens.

   (b) City-list Filtering

   We also observed that several tuples being generated from our system were not meaningful as the city name were too generic words in English language. Some of the most frequent cities in our results

Mosque Name	Ambiguity Reason	Example Website
Finsbury park Mosque	...*in London*	`http://www.historycommons.org/entity.jsp?entity=finsbury_park_mosque_1`
Lal Masjid	...*in Pakistan*	`http://imran.com/media/blog/labels/Mosque.html`
Babri Masjid	...*in India*	`http://medlibrary.org/medwiki/Mosque`
Less popular mosque	...*in news stories*	`http://www.france24.com/en/20090528-iran-mosque-suicide-attack-kills-zahedan` `http://www.globaltv.com/globaltv/winnipeg/Suicide+bomb+kills+Pakistani+mosque/1434854/story.html`
Less popular mosque	...*in community sites*	`http://www.cybercity-online.net/Pakistan/html/shrines_tombs___mosques_in_pak.html`

Figure 36: Example results illustrating AF vs. non-AF ambiguity.

included *monday, march, or, top, main, want, taliban, taleban, hindu, burma, china, masjid, park* since there are cities in NGA Gazetteer for Afghanistan with these names. An ideal solution here would be to disambiguate if occurrence of these terms; however, in our current implementation, we simply compiled a "stop-list" of city names, like above, that are more likely to be used as generic term rather than as a city name, and removed the tuples for which city name belongs to this stop-list.

3.3.4 Large-scale Crawling

Having designed our technical components, to deliver the final system, we need to deploy our text extractor onto large scale of Web corpus. Our objective here is to prepare a corpus containing pages relevant to information about mosques in Afghanistan. We took following two methods to obtain these pages:

1. Focus Crawling. We started by obtaining the top 1000 results from MSN Live search engine for the general query "mosques in Afghanistan." We dispatched our focus crawling program to crawl pages at depth of 1, 2 and 3 from these initial URLs.

2. Direct Discovery. In addition to the indirect approach of deeper crawling from initial relevant pages in above method, we also attempted to discover the relevant pages directly. We used NGA Gazetteer for Afghanistan to drive our direct discovery of relevant pages. For every city in the gazetteer, we queried MSN Live search engine with the query patterns of "+mosque #city Afghanistan," and "+masjid #city Afghanistan." For each query, we obtained maximum up to top 1000 results. Note that, for many of these queries we had fewer than 1000 results available; for these cases, we obtained all the available results. Together, this method provided us a pretty extensive list of pages having mosque information relevant to any city in Afghanistan.

3.4 Merging Mosque Features

Upon the raw results extracted from Text Extractor module, we need to apply *GeoMerging* step to group the raw records referring to same mosque feature into a merged record. The final score for the merged results are computed as the sum of the scores of individual raw mosque records, similar to our implementation for previous *GeoEngine* tasks. During the merging step, our system needs to handle several forms of ambiguities, as described below.

3.4.1 Ambiguity due to AF vs. non-AF context

While our crawling attempts to collect the pages relevant to mosque information in Afghanistan, we still include many pages that are ambiguous. As a result, many of the tuples obtained from our text extractor were not in Afghanistan. Figure 36 shows some illustrative results that we need to filter out, as they are not in Afghanistan. For example, one of the most frequently extracted mosque was "Finsbury park Mosque" which is London. It is mentioned in many pages on the Web, due to its extensive media coverage in recent times. However, our text extractor incorrectly assembles this mosque name with a city in Afghanistan mentioned on the same page. Some other examples include "Lal Masjid" in Pakistan, and "Babri Masjid" in India. We found that these type of ambiguous tuples come from variety of pages–from cultural pages, news articles, *etc.*

To filter out these results, we extract additional context in our text extractor, that help us in disambiguate if the content of the page is really about Afghanistan or not.

NonAFScore For each page, we computed a score that indicates if the content of the page is **not** about Afghanistan. The score is computed by counting the number of mentions of terms in "nonAF list," which is a manually compiled list of foreign countries, *e.g.*, "USA," "Pakistan," "Oman," "Sudan," "Iran," *etc.* and some popular foreign cities, *e.g.*, "Mecca," "Islamabad," *etc.* We also included "personification" variants for some of the location names, *e.g.*, "Pakistani," "Iraqi," *etc.* It is important to note that the list is not exhaustive–we included only those locations that are likely to make our mosque tuple extraction ambiguous. For example, we did not include country "Brazil" in our list; consider a page which mentions "Brazil," a mosque name, and also, a city in Afghanistan–quite likely, the mosque name mentioned in that page is really associated with that city in Afghanistan, rather than with Brazil.

AFScore Likewise, for each page, we computed a score that indicates how likely the content of the page is about Afghanistan, by counting the number of mentions of terms in "AF list." This list contains the country name "Afghanistan" and its "personification" variants, *e.g.*, "Afghan," "Afghani," and "Afghanistani."

Resolving AF vs. NonAF Ambiguity Based on the two scores–AFScore and the NonAFScore–we determined if the tuples extracted from a page are about Afghanistan or not. If either $NonAFScore > 5$, or $AFScore < NonAFScore$, then we considered the page to be not about Afghanistan, and filtered the tuples extracted from these pages.

We did not rigorously evaluate these filtering conditions; however, in our experience, they seemed to work well, except for a few situations where our criteria may incorrectly filter out pages that are about Afghanistan such as (a) If the page has a story connecting to many countries, *e.g.*, a news story about an incidence in a mosque in Afghanistan, citing comments from officials from USA and Pakistan (`http://www.earthtimes.org/articles/show/269069,insurgents-rocket-hits-afghan-mosque-killing-five.html`), or (b) If the page includes mentions of "foreign" locations in sections not related to the actual story, *e.g.*, a news story which references only locations in Afghanistan; however, the page includes a selection box containing list of all countries (`http://www.irinnews.org/report.aspx?reportid=28652`).

3.4.2 Ambiguity due to different features with same city name

The second ambiguity arises when merging raw records into merged results of unique mosque features. Our text extractor returns tuples of the format (#mosque-name, #city); now, in our reference NGA Gazetteer, there could be multiple features with same city name. So, we cannot merge the raw records, by only matching the value of #city.

Our solution to resolving this form of ambiguity is inspired by our experience of addressing this challenge during Phase I problem of finding population. When extracting the mosque tuples, in our text extractors, we augment the tuple with additional *context* information. More specifically, to understand the context of a city, we extract the other city names in that page. When merging two raw records, we compare the context of their co-occurring city names. The records can be merged together, only if the overlap between the two city list are greater than a threshold. In our implementation, as threshold, we require that at least 5 cities, as absolute count, and at least 20% of the entries, as fractional overlap, should be common between the city-list context of the two raw records.

3.4.3 Ambiguity due to variants in mosque names

Another challenge in merging raw records from text extractor is the variation in the names of same geo-feature. For example, the same mosque feature may be referenced with multiple names, *e.g.*, "Masjid Sabz" and "Green Mosque" both refer to the same mosque in the city of Balkh, Afghanistan.

To handle the variations in mosque names, we developed customized functions to judge if two mosque names are similar. *First*, we prepare a list of "stop-words" used in mosque names including popular tokens, *e.g.*, "the," "a," "of," *etc.*, and common nouns representing synonyms of mosque, *e.g.*, "mosque," "masjid," "shrine," *etc.*. This first step can be considered as "stemming" for mosque names. *Second*, upon the key terms remaining after stemming, we compute the token level *edit-distance*. We use the inverse of this edit distance as the measure of similarity between mosque names. If the similarity score is above certain threshold, we consider the two names as the variant of same mosque.

3.5 Performance Evaluation

To evaluate "Afghanistan Mosque Discovery" system, we performed precision and recall analysis; identified possible reasons for errors and proposed solutions. Additionally, we studied the distribution of merged records to further understand domain characteristics that we observed.

3.5.1 Precision Analysis - Reasons for Errors and Possible Improvements

From the 3816 merged results, we sampled the top 60 records, and manually evaluated each, in order to assess the approach and to gain insight of potential improvement. Therefore, for each record, we not only checked whether the association is correct but also identify the reasons of errors. Figure 37 shows this categorization of the results. We identified nine categories: $E1$–$E6$ are *wrong* associations of various reasons, $A1$ and $A2$ are *acceptable* results with minor error, and G) is *good* results. Overall, the *precision* measuring those acceptable and good results is 48.4%—and we observed valuable insight for improving the results. In the following, we discuss each category, and how the results could be improved.

E1: Mosque name and city name were overlapping. In Text Extractor, we currently allow extracting #city and #mosque names from an overlapping piece of text, resulting in errors when the city mentioned in a mosque name is not the actual city. For example, from the text "... Ibrahim Shah Baba Shrine ...," we extracted #mosque as "Ibrahim Shah Baba Shrine" and #city as Baba, since Baba is in the list of city names to recognize— however, the correct city of this mosque is Qalechah. As another example, we extracted (Haji Yacob Mosque, Haji), which should be (Haji Yacob Mosque, Kabul) instead.

How to fix this type of errors? In extraction, we can require that a #city does not get extracted from *part* of a #mosque—In general, we should extract a segment of text as either #mosque or #city, but not both.

E2: Mosque names in forward context were ignored. We currently discover #mosque name using backward context only, *i.e.*, where the key mosque terms like "Mosque" or "Shrine" appear at the end of a name, such as "Haji Yacob Mosque." Thus, names that starts with the key terms, such as "Mosque of the Cloak of the Prophet Mohammed," where the name appears in the forward context of the keyword "Mosque," are not extracted properly. (Another example: "Shrine of Hazart Ali.") In such cases, we extracted it as only "Mosque" or "$*$ Mosque," where $*$ is the word immediately preceding "Mosque."

How to fix this type of errors? We need to extend our mosque name extraction to look for mosque names not only in the backward context of key terms, but also in their forward context.

Category	Description/Reasons of Errors	#Cases	Percentage
E1	City name should not overlap with mosque name	8	13.3%
E2	Mosque name should be extracted using forward context	1	1.7%
E3	City name confused with a foreign city	2	3.3%
E4	Should not associate with non-informative segments	13	21.6%
E5	Should section text to infer association	2	3.3%
E6	City name is confused with person name	5	8.3%
	Need-Improvement Total	**31**	**51.6%**
A1	Acceptable: Part of the mosque name is not right	2	3.3%
A2	Acceptable: Wrong association, but low rank	1	1.6%
G	Good association	26	43.3%
	Acceptable/Good Total	**29**	**48.4%**

Figure 37: Evaluation of mosque discovery: over top 60 results.

E3: Foreign cities were not recognized. We do not currently distinguish cities of the same names but in different countries– the *geo disambiguation* problem. As we discussed in the Phase I project, when we studied finding populations for Benin features, a name can refer to multiple locations; *e.g.*, Gando is a name for 12 UFIs. Thus, when searching for a specific location, say Gando with UFI=-1333543, *GeoEngine* cannot simply match by the name alone; rather, it must take measures to disambiguate whether the name refers to the particular location we seek. This *toponym resolution* problem has been actively studied (*e.g.*, [42, 60, 73, 76, 77, 78, 79, 82, 86, 89, 101]) for various settings such as video, speech, text, and the Web.

How to fix this type of errors? While we studied the same geo disambiguation problem in the context of augmenting an existing gazetteer (adding Benin populations to the NGA Gazetteer), now, we do not have a gazetteer to start with for the various features (*e.g.*, the different "Gando") that may be ambiguous. That is, since our #city extraction is only based on cities in Afghanistan, we are current unable to distinguish those city names that also appear in other countries. As a solution, we will add city gazetteer information for other Arabic countries, in order to distinguish Afghanistan cities.

E4: Mosque and city were associated across separate regions. A page is usually divided into multiple regions (navigation, content, advertisement), and each regions further divided into different sub-regions of *information units* like sentences, records, and links. In this category, we incorrectly associated a #mosque with #city, while they appeared in unrelated regions of the page. For example, Figure 38 shows a snippet of navigation links section at the bottom of the page on *wikipedia* about "Abdul Rahman Mosque" (`http://en.wikipedia.org/wiki/Abdul_Rahman_Mosque`). In this snippet, we extracted #mosque from one navigation link (as marked in the circle) and #city from the following link (as marked in the rectangle). Clearly, while close, the information across two distinct navigation links should not be associated—*i.e.*, proximity is not a *sufficient* condition for tuple association.

How to fix this type of errors? In forming associations, we should consider proximity only *within* a region of information, and not across two regions that are independent, even when they appear close to one another in the overall page layout. First, we will need to segment pages into sub-units [52, 53]; and then, apply our tuple association algorithm on the content within each sub-unit of a page.

E5: Long range associations were not assembled. In this category, we failed to associate a #mosque and a #city entity because they were not close to each other. While we have focused on text proximity as a primary means for assembling tuples, we observed that some prominent associations are not limited to proximity—*i.e.*, we are missing "long range" associations. Thus, proximity is not a *necessary* condition for tuple association.

How to fix this type of errors? It is important to note that *proximity* is simply a measure of the "scope" of information—If a #mosque is near a #city in terms of textual distance, we presume that the former is within the scope of the latter (or vice versa). In general, we shall consider the "scoping" of text, *i.e.*, the range of influence—If an information entity is in the scope of another, we can associate them as a tuple, while not necessarily in textual proximity to each other. For example, information appearing in the title of a page describes the whole page, and thus it naturally has the entire page as scope. We should generalize proximity to account for such scoping in order to assemble long range associations.

E6: Person mentions were mistaken as city names. There are many names commonly used for both person and location. This type of errors thus resulted from wrongly extracting person name mentions as #city (which were then associated with some #mosque). For example, we assembled tuple (Sect's Mosque, Yusuf); however, this Yusuf was referring a person in the extracted text, although it is also a city name in our dictionary for extracting cities. Another example is the merged record number 26, where "Tamim-E-Ansar Mosque" is group with *Qari* (a city name), while in fact Qari is a person's name, as in "Mullah Qari Mushtaq".

How to fix this type of errors? We need to distinguish these two entity types so that we do not group mosque name with person name (even when another city name is nearby in the surrounding).

Figure 38: Example snippet of a page where #mosque and #city appear in navigation links.

Mosque Name	City Name	#Tuples
Abdul Rahman Mosque	Kabul	3
Friday Mosque of Herat	Herat	41
Green Mosque	Balkh	4
Haji Piyada	Balkh	1
Id Gah Mosque	Kabul	5
Khost Mosque	Khost	2
Khwaja 'Abd Allah Ansari Shrine	Herat	2
Lashkar Gah Mosque	Lashkar Gah	1
Mosque of the Cloak of the Prophet Mohammed	Kandahar	0
Mosque of the Hair of the Prophet Mohammed	Kandahar	0
Pul-e Khishti Mosque	Kabul	3
Shah-Do Shamshira Mosque	Kabul	5
Shrine of Hazrat Ali	Mazar	6

Figure 39: Recall study of mosque tuple assembly.

A1: Part of the mosque name is not right. There are many cases where mosque name is extracted wrongly. In some cases, mosque name extraction module spanned more tokens than desired. In other cases, it missed to include some of the words into the mosque's name. For example, in merged record number 44 (`http://www.clovekvtisni.cz/index2en.php?parent=&sid=&id=402&idGallery=21`), "Author Ladislav Kudlacek Blue Mosque" is identified as mosque name. However, the correct name should have been "Blue Mosque" in this case.

How to fix this type of errors? We need to improve our mosque name extraction by adding more features to identify tokens that should act as boundary of mosque names, when going backward (or forward) in context from key terms. Specifically, we can consider other information such as case-sensitivity and special symbol, beyond our current implementation of using stop word lists, punctuations, and digits as name boundaries. In the example mentioned above, we can use HTML *br* tag, to act as a potential entity boundary and extract "Blue Mosque" as mosque name correctly.

A2: Wrong association, but lower rank than correct association. In current system, we showed all the potentially merged mosques in ranked order. As a result, the same mosque could be associated with different cities. For example, in merged mosques number 55 and 40, Hanzala Mosque is associated with Shahr and Kabul respectively. The correct association in this case is the one with lower rank.

How to fix this type of errors? One possible method to get correct results is to prune out same-mosque-name records with lower rank. This method, however, is quite extreme and sensitive in the sense that it might delete good associations. Another possible improvement is to group results with common mosque name together and present all of them to system operator for close inspection.

3.5.2 Recall Analysis - Coverage of Tuples Discovered

In order to perform recall analysis, we decided to match our results with information from a trusted source to see what percentage of data we can cover. However, Afghanistan mosque information has never been formally assembled together before. The closest effort is from *IslamicFinder*, and it has only 10 records, out of them only 7 contain mosque information. Therefore, we picked *Wikipedia*, a source with information of 13 mosques, as ground truth to compare against. Given a mosque name in these 13 mosques, we checked if there existed a correct association in merged results. The results are summarized in Figure 39. The *#Tuples* column indicates how many merged tuples cover the corresponding association of mosque name and city name.

As shown in Figure 39, the final recall is 11/13, about 84.62%. For the two cases where there's no covering tuples, it is due to the fact that our current implementation of mosque name extraction is not capable of recognizing these mosque names. As discussed earlier in case E2 in precision analysis, we will extend our mosque name extraction to include not only backward context, but also the forward context of the key terms.

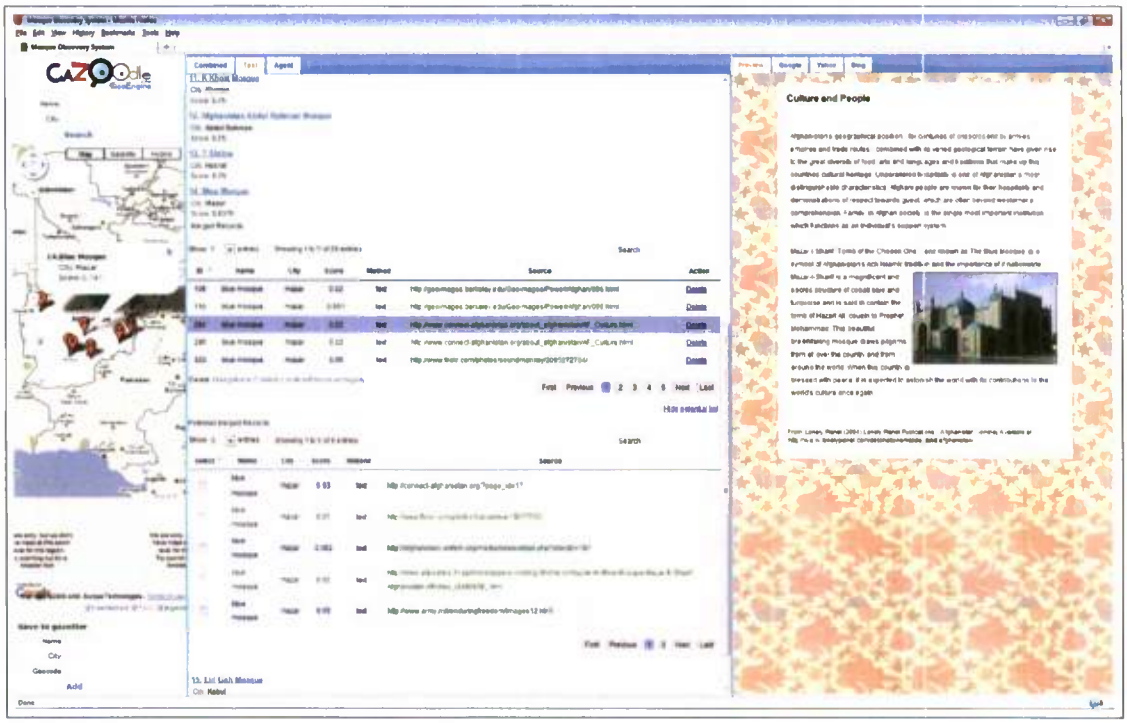


Figure 40: Illustration of user assistance in correcting record merging.

3.6 User in the loop

It is important to note that, the problem of geo-merging is inherently more difficult for the task of finding Mosques in Afghanistan, when compared to the problem of finding Hospitals in Chicago. In case of Afghanistan, the geo-coordinates of the extracted tuples are available at the granularity of the city names. In contrast, for Chicago tasks, we had full street addresses available. And so, even if the names of the hospitals were totally different, we could still merge the records based on their geo-coordinates. For example, using our algorithm, we cannot merge the two mosque tuples with mosque names as "Masjid Sabz" and "Green Mosque," since we only know they are both in Balkh city; we do not know their specific location inside the Balkh city. In contrast, for Hospitals in Chicago task, we could merge hospitals with names as "Cook County Hospital" and "John H. Stroger Memorial Hospital," since their full street addresses were exactly the same.

This motivates us to involve analysts who are using our operation console in the loop of discovering mosques in Afghanistan, more than in the discovery of hospitals in Chicago. Besides showing the confident merges, as determined by the threshold conditions for mosque geo-integration, we also prepare "potential" merges. Alongside each merged result, we also list all the other raw records, that fall short of our threshold conditions; however, still likely to be candidates for merging. Analysts can then inspect these potential merges and decide if these raw records should be merged as well.

3.6.1 Operation Console

Similar to our prior systems, our target user of the system is an analyst who can browse and inspect through the candidate results, and make final decisions of adding the new features to the gazetteers. Therefore, our design of the *Mosque in Afghanistan* system, as shown in Figure 41, also uses similar layout in 3 vertical panels: a) The left panel is for querying and browsing the discovered mosque features. b) The middle panel is for showing the mosque results matching search conditions. c) The right panel is for browsing and examining each related link from the results.

First, the query and browsing panel allows users to easily navigate through different mosque features discovered by our system. The panel consists of 3 horizontal sections. At the top, it provides a query form for searching

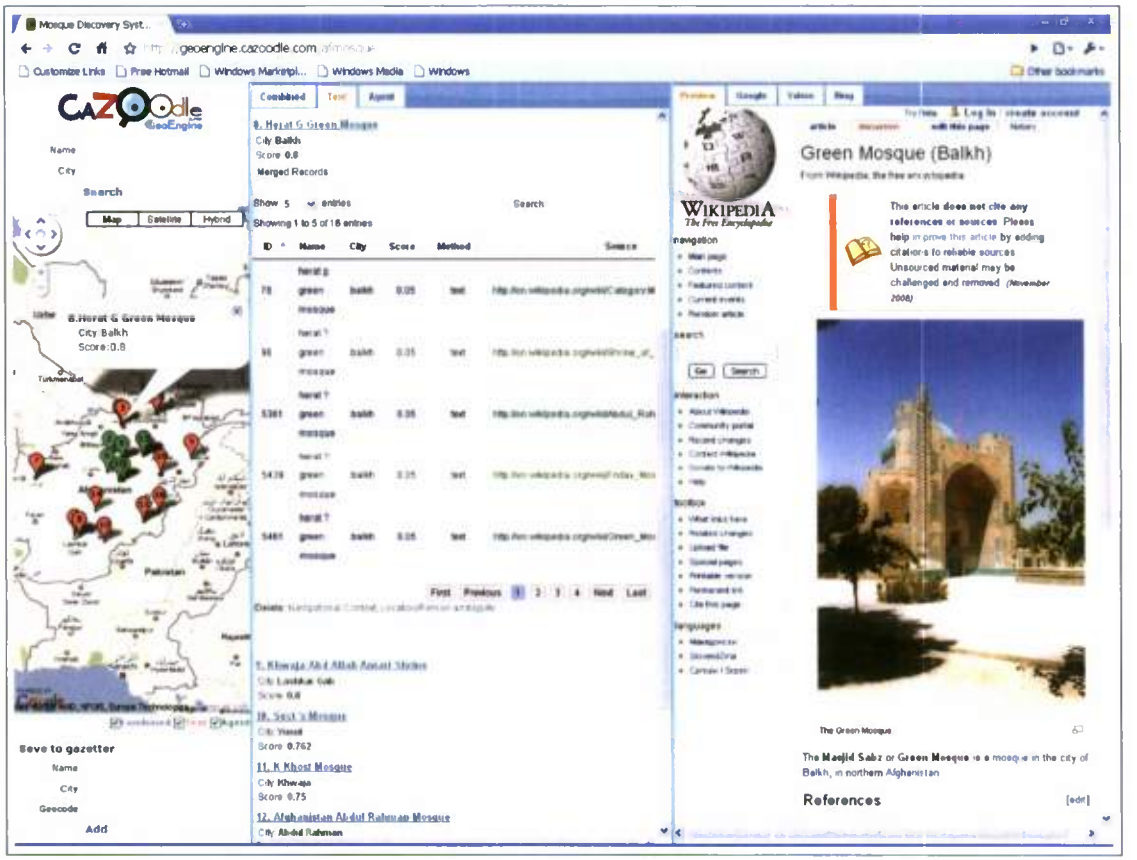


Figure 41: The operation console of *GeoEngine* for discovery of Mosque in Afghanistan.

mosque by mosque-name or by city. The middle section provides integration with a map, for users to visualize the location of the matching mosques. The bottom section lets users save the browsed entries to the final gazetteer.

Second, the middle panel displays the mosque features that match a current query (as specified in the left panel). There are three tabs, which displays agent results, text results, and combined results. However, as we only focus our technique on text sources, only the Text tab has information about mosques that we integrate from multiple sources together presented in a ranked order of score. Users can click on any result to see more detailed information in the right browsing panel. Furthermore, we provide users with two new functions in this interface: (a) Ability to see the potential merges and merge them into the displayed group, and (b) Remove incorrectly merged entries, and to search for other raw records sharing similar context.

Third, the right panel for Web browsing lets analysts browse and inspect the context where the information appears, in order to draw conclusion about accuracy of results. Users can click on the URL hyperlinks in each result (in the middle panel) to navigate to that URL in the "Preview" tab.

Overall, the three panels together provide an integrated operation console, much like what we also offered in the Phase I prototype for finding population and in the Phase I option task of discovery of Hospitals in Chicago–for analysts to browse through the discovered hospital features, make judgments on the accuracy of results, and import qualified records into the gazetteer.

3.6.2 User assistance in correcting record merging.

We recognize that the problem of mosque discovery is more challenging than the *GeoEngine* applications we studied earlier. Our precision here is 48%, while recall is 85%. Besides the areas of improvements that we identified, we also believe the users of the system need to participate more actively–not only in making final decision, but also in giving feedback to system for improving accuracy.

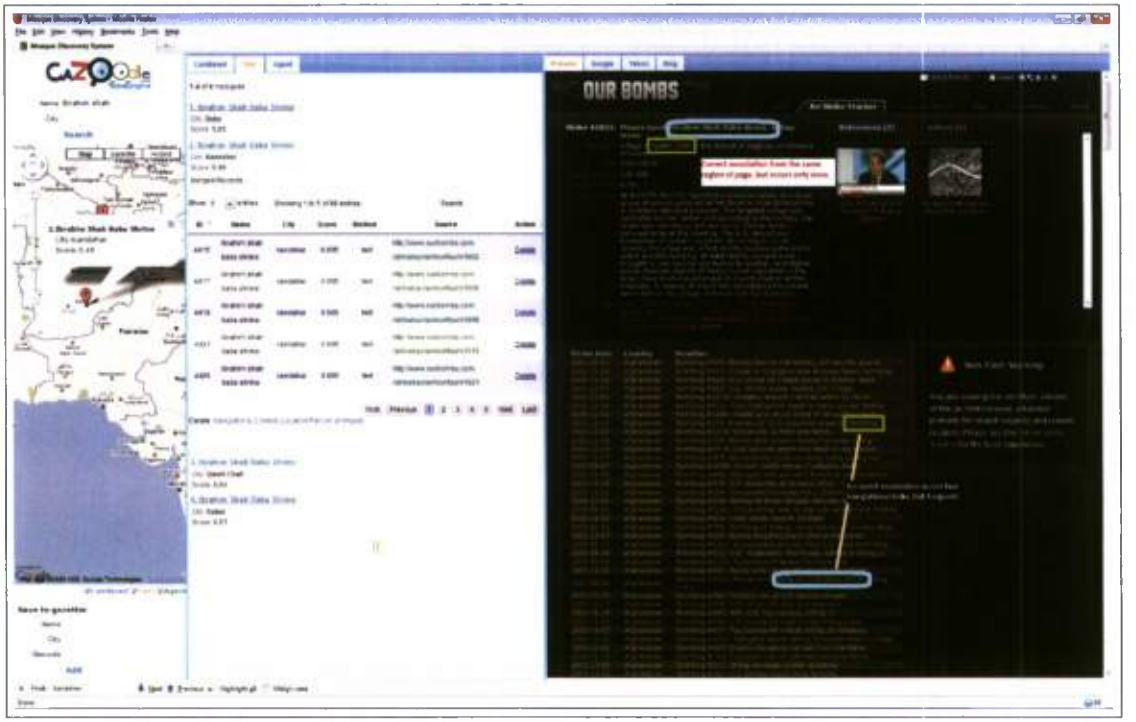


Figure 42: Errors due to extraction from unrelated regions of the page.

For merging raw records, we compute similarity between them–based on similarity of mosque name, city name, and the context of other city names mentioned in that page. Unlike the hospital task, where we had full address and precise geocoordinates available, here, our merging algorithm has to rely only on the names of the cities.

To overcome the inherent difficulty in merging, we will provide more functions in our interface for users to inspect and make changes in the merging results generated by the system.

1. Approve Potential Merges For each merged result, we return two sets of raw records–the first set of *confident merges*, and the second set of less confident, *i.e.*, *potential merges*. Figure 40 shows two sections for the mosque result ("Blue Mosque", "Mazar"). The raw records in potential merges section may have differences in mosque name, or city name, or more likely, in the context of the list of cities in the page where that raw record was extracted from. Users can click on the source link to inspect the page content in the "preview" tab. If the raw record indeed matches the merged record, users can use check-boxes to select and move that raw record from the potential merges section to the confident merges section.

2. Delete Confident Merges Quite possibly, our merging algorithm may make mistake in erroneously putting some raw record in the confident merges section. We provide users a "delete" function to remove these raw records from the group, before inserting this saving this merged record into gazetteer.

3. Raw Records Filtering In general, the number of raw records appearing in confident merges or in the potential merges section could be large. For example, for the screenshot in Figure 40, there are 29 raw records in confident merges section, and 6 raw records in potential merges section. To facilitate browsing through these raw records, we provide client-side search. Users can type any keyword in the keyword box (*e.g.*, "news"), and the results will be filtered to raw records matching these keywords. In our current implementation, the keyword search is supported on mosque name, city name, and the tokens in URL. This search can later be extended to also provide filtering on type of content (*e.g.*, images, videos, news, text, *etc.*.) or metadata properties (*e.g.*, title of the page, meta-keywords, meta-description, *etc.*) and full page content.

Figure 43: User feedback for detecting unrelated regions of the page.

3.6.3 User feedback for improving text extraction

Many of the errors we identified in our analysis require us to improve our text extraction modules. Some of these errors are context dependent, and will require us to compile rules specific to the target application. We envision to involve users of our system in learning these rules. In the current implementation, we "simulate" such a learning component for two specific types of errors–extraction across unrelated regions of the page (E4), and ambiguity between person vs. location names (E6).

1. Avoiding extraction across unrelated regions Our current text extraction considers the content of a page as one single string, and applies recognition and assembling of mosque name and city names anywhere in that string. As a result, often our results include mosque tuples that are associating entities from unrelated regions of the Web page. Consider, for example, the results from our system shown in Figure 42 for mosque name "Ibrahim Shah Baba Shrine." The correct result here is ranked at position 3, obtained from the news story about this mosque (top of the page, shown in the preview panel in Figure 42). The first result is erroneous due to error type E1, and will be fixed by avoiding mosque name and city name to be overlapping. The second result is erroneous as the extraction occurs across the unrelated region–from the anchor text of two hyperlinks pointing to independent news stories. The anchor text of one of the hyperlink mentions city name "Kandahar," while the other mentions the mosque name "Ibrahim Shah Baba Shrine."

To avoid these errors, our system needs to learn these templates, or patterns, *e.g.*, list of hyperlinks with structural similarity represent unrelated text. We "simulated" this learning, by having user provide feedback to the system when they come across such erroneous extraction. As shown in Figure 43, user can select a mosque tuple, and then click on the "Navigational Context" function to see all the mosque tuples extracted from the same "context" as the selected tuple. User can review the list, and then choose to delete all the tuples. System will then set the scores of all these raw records to 0. In the current implementation, we look at source name and mosque name as context, which means, even the correct result ("Ibrahim Shah Baba Shrine", "Qaleh Chah") will be deleted. In potential future development, we might refine the context to look for tokens and other HTML features in the surrounding context of the selected tuple.

2. Resolving ambiguity between location and person name There may be a problem distinguishing

Figure 44: User assistance in resolving ambiguity between location and person name.

person's name a from location name, when the same name can be used as both the name of a person as well as of a location. The general approach for addressing this problem is to learn contextual rules based on labeled examples [61]. Whenever users come across a result where the extracted city name actually represents the name of a person, they can provide the feedback to the system. Consider, for example, the mosque result ("Sect's mosque", "Yusuf"), as shown in Figure 44. While "Yusuf" is a city name in Afghanistan, from the page in the preview panel, we can see that here it refers to the name of the person ("Mr. Yusuf", or "Yusuf's death", "Yusuf was"). User can click on the "location/person ambiguity" function, our system will recognize the contextual pattern, and return a list of other occurrences matching that context. In the current implementation, we simply look for all the occurrences of "Yusuf" as the city in that specific source. In potential future development, this function might return all the city names we extracted in the matching context, *e.g.*, "Mr. #city" or "#city was." Users could inspect the list of other results sharing the same context, and click on the delete button. This would provide feedback to the system to add a new contextual rule to resolve the ambiguity between a location and a person name.

3.7 Text Extraction Technique Improvements

We implemented several enhancements in our text extraction module, based on our observation of its performance. Earlier, we evaluated the performance of our mosque discovery system. The error cases of our system were categorized into different reasons for errors, as previously summarized in Figure 37. Specifically, we added the following enhancements:

3.7.1 Backward-context name extraction improvement

In our previous implementation of text extraction, we extracted names of the mosques by first identifying "key terms" (*e.g.*, mosque, masjid, shrine, *etc.*) in a text, and then moving backwards in text content until a "name separator" (*e.g.*, verbs, sentence boundaries, *etc.*) was found. The sequence of terms between the name separator and the key terms would be considered as the name of the mosque.

We observed that our name extraction results were not quite accurate. So, we added enhancements of:

Comparison	Previous implementation	Enhanced implementation
4 same results	blue mosque imam ali mosque blue mosque shamshira mosque	blue mosque imam ali mosque blue mosque shamshira mosque
8 names cleaned	afghanistan abdul rahman mosque khwaja abd allah ansari shrine herat ? green mosque piyada ? id gah mosque prophet ? pul-e khishti mosque ? friday mosque ? khost mosque ? lashkar gah mosque	abdul rahman mosque abd allah ansari shrine green mosque id gah mosque pul-e khishti mosque friday mosque khost mosque lashkar gah mosque
3 names pruned	? mosque prophet mohammed ? mosque ? shrine	— — —
9 new results		mosque of herat mosque of the cloak mosque of the hair shrine of hazrat ali shrine of hazrat ali basic shrine of hazrat ali shrine of ali shrine of kurush shrine of hazrat ali

Figure 45: Comparison of mosque names extracted by the previous implementation and the enhanced implementation from an example Web page.

1. Name cleaning: We cleaned the name of the mosque to remove special characters.

2. Pseudo-adjective name separators: We added new rules for determining if a text token is a name separator. Previously, we had a long list of "stop-words." This list is difficult to maintain or compile. As an enhancement, we observed that many of our incorrect name extractions were the result of an adjective included in the name of the mosque. So, we added new rules that would consider tokens ending in "-ed" or "-ing" as name separators.

1 Forward-context name extraction

Our previous implementation of name extraction, as described above, used to go backwards in context from the tokens matching key terms. As our analysis of different categories of errors showed, our previous implementation missed in extracting mosque names that required forward context, *e.g.*, "Shrine of Khwaja Moin-u-din." We used the same rules for determining whether a token is a name separator, and adopted our backward-context name extraction to also trace in a forward direction with regard to text segment.

2 Tuple association improvement

In our previous implementation, we were returning all pairs of the mosque names and the city names that were extracted from a Web page as candidate results. These pairs were scored based on the proximity of the two entities in the text segment. While this scoring scheme worked well, we found that, upon aggregation across sources, some of the less likely associations were ranked higher overall. As we see from the difference between Figures 37 and 45, one of the key areas of improvement was related to pruning out meaningless associations.

We modified our tuple association algorithm as follows:

1. Disallow overlapping city name and mosque name: We found that sometimes one of the tokens in the sequence of tokens comprising a mosque name may also represent the name of a city in Afghanistan. For example, consider the mosque name "Baba Khan Shrine": the token "baba" also represents a city in Afghanistan.

Site	Backward					Forward				
	Labeled	Extracted	Correct	Precision	Recall	Labeled	Extracted	Correct	Precision	Recall
W1	8	8	8	1	1	3	3	1	0.33	0.33
W2	1	1	1	1	1	0	0		NA	NA
W3	2	2	2	1	1	0	0		NA	NA
W4	1	1	1	1	1	0	1		0	NA
W5	1	1	1	1	1	1	1	1	1	1
T1	0	1	0	0	NA	0	1		0	NA
T2	0	2	0	0	NA	0	0		NA	NA
T3	1	1	1	1	1	0	0		NA	NA
T4	0	0	0	NA	NA	0	0		NA	NA
T5	1	1	1	1	1	0	0		NA	NA
T6	3	2	2	1.00	0.67	0	0		NA	NA
T7	0	0	0	NA	NA	0	0		NA	NA
T8	0	0	0	NA	NA	1	0		0	0
T9	1	3	1	0.33	1	0	0		NA	NA
T10	1	1	1	1	1	0	0		NA	NA
All	20	24	19	0.79	0.95	5	6	2	0.33	0.4

Figure 46: Evaluation of the accuracy of the enhanced implementation of the mosque name extraction.

Context	Popularity	Precision	Recall
Backward	80%	80%	95%
Forward	20%	33%	40%
Overall	100%	70%	84%

Figure 47: Summary of the accuracy of the mosque name extraction.

Our previous implementation included this pair as one of the candidate tuples, since it was simply returning all pairs with proximity scores. So we next added a new rule to our tuple association function which would prune out such associations.

2. Tightest binding of the city name and mosque name: We also found that there may be multiple cities mentioned in the proximity of a mosque name. Our previous implementation involved associating all pairs of mosque tuples. We added an enhancement that would prune out the associations that did not represent the tightest binding. With this new rule, all the output tuples represented the tightest binding associations, *i.e.*, the text content found between the occurrence of the mosque name and the city name did not include any other mosque name or city name.

3.7.2 Text Extraction Performance Evaluation

We first compared the performance of our previous implementation with the new implementation on many Web pages and saw significant improvement. For illustration, consider the Web page on *Wikipedia* at this URL: `http://en.wikipedia.org/wiki/Shrine_of_Hazrat_Ali`. We compared the list of mosque names extracted from our previous implementation and the new system after the above enhancements were added, as shown in Figure 45. As the figure shows, the first 4 mosques names were the same in the two implementations. We found that there were 8 mosque names for which our new implementation extracted a cleaner name. Next, we found that there were 3 mosque names in the output of the old system that did not represent any mosque–all of these 3 mosque names were pruned out in the new implementation. Finally, we found that the new implementation returned a number of mosque names that the old system did not produce, *e.g.*, "mosque of herat," "shrine of hazrat ali," *etc.* These mosque names were extracted using the latest enhancement of extracting the mosque names in the forward context. On this example page, a total of 9 mosque names were extracted by the new implementation using forward context enhancement.

Result Quality	Ratio
Good results	65%
Name Extraction Improvement Needed (#10, #12, #15)	15%
Web Page Segmentation Needed (#4, #5, #7, #14)	20%

Figure 48: Evaluation of the performance of the end-to-end system.

Next, we systematically evaluated the performance of our mosque name extraction technique, using the metrics of precision and recall, on a sample of 15 Web pages. We recorded our evaluation results in Figure 46. We prepared the sample of the 15 pages as a mixture of 5 Web pages from *Wikipedia* (marked as W1 - W5 in the figure), and the other 10 from other text sources (marked as T1 - T10). We manually inspected each of the Web pages to identify all the mosque names mentioned in these sample pages. We separately recorded the performance of our mosque name extraction in the backward vs. forward context. For example, for the Web page W1, we found 8 mosque names in the backward context and 3 in the forward context. For the same Web page, our mosque name extraction returned 8 results in the backward context, and 3 results in the forward context. We then inspected these results to determine how many were correct; we found that for the backward context, all 8 results were correct, while for the forward context, only 1 out of 3 results was correct. Thus, for the Web page W1, precision and recall for the backward context were both 100%, while in the forward context, they were both 33%.

We summarized these performance results in Figure 47. After manually inspecting all 15 Web pages, we found a total of 25 mosque names, of which 20 were found in the backward context, while 5 were found in the forward context. Thus, the backward context seemed to be more popular when naming mosques. Our text extraction technique seemed to perform better for the backward context than for the forward context. For the backward context, we found that precision scored 80% and recall scored 95%, while for the forward context, precision was only 33% and recall 40%. The overall combined performance, including mosque names in either of the contexts, was 70% on precision and 84% on recall.

3.7.3 End-to-end System Performance Evaluation

We incorporated our new implementation into an end-to-end mosque discovery system, publicly available at `http://geoengine.cazoodle.com/afmosque/`. A quick inspection of the results shows that the quality of results were significantly improved over the previous implementation, which can be accessed at the following URL: `http://geoengine.cazoodle.com/afmosquev1`. The accuracy of the previous implementation was only 45%, as we found that several of even the top-ranked results were incorrect, *e.g.*, the top result of the previous implementation was for the city name "Baba" which overlapped with the sequence of tokens comprising the mosque name "Ibrahim Shah Baba Shrine."

For a systematic evaluation of the performance of the end-to-end system, we inspected the top 20 mosque results of our new implementation. As summarized in Figure 48, we found that the accuracy of the new implementation is now 65%, which is much higher than the 45% accuracy of the previous implementation.

We again categorized these erroneous results into different categories of reasons. We found that 15% of the errors resulted from incorrect mosque names, indicating that our mosque name extraction still needs improvement. The remainder of the 20% results were incorrect due to erroneous tuple association. More specifically, for these results, our algorithm is associating the city name and the mosque name from unrelated sections of the Web page, indicating the need for segmenting the Web page into coherent units prior to dispatching the pages to text extraction modules.

3.7.4 Enhancing Text Extraction with the NLP Techniques

As our second task, we studied how we can incorporate Natural Language Parsing (NLP) techniques to enhance the accuracy of our text extraction techniques. As we had previously concluded, this would be one of the key areas of enhancements in our handling of text sources.

To begin with, we surveyed existing literature for part of speech (POS) tagging, and decided to use Stanford's POS tagger for our work [95]. The goal of POS tagging is to determine the correct part of speech for each word in the text. As illustrated in Figure 49, Stanford's POS tagger [95] uses a cyclic dependency network. In contrast to

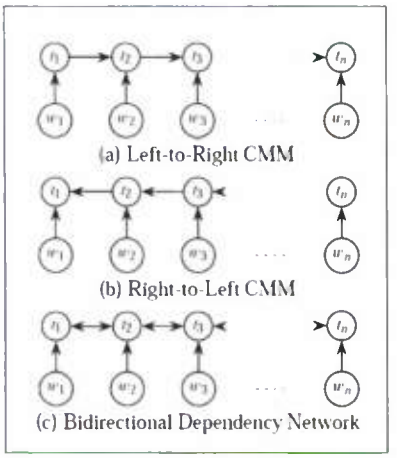


Figure 49: Cyclic Dependency Network as used in Stanford's POS tagger (image obtained from [95]).

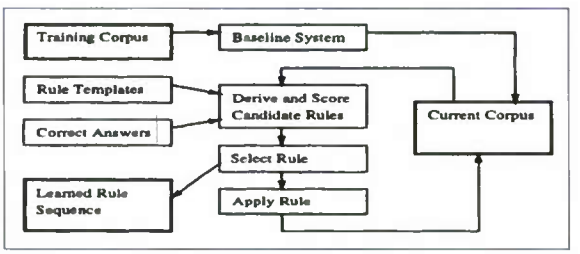


Figure 50: Transformation-based learning used for NP Chunking (image obtained from [84]).

some of the traditional approaches that use *unidirectional* inferencing [49, 59, 85] for sequence tagging, the cyclic dependency network captures dependencies in both directions. Such joint inferencing has been shown to yield superior performance.

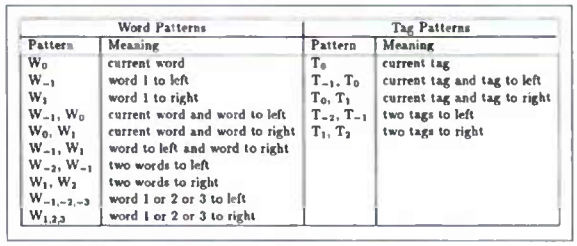
We used these POS tags, in turn, as input for *text chunking*, which will identify all the *noun phrases* (NP) in a given text. Introduced in 1991 [41], text chunking, and specifically NP chunking, has also been an active area of research within the NLP community [48, 59, 75, 98]. We used the text chunking technique [84], inspired by transformation-based learning, which was originally used for POS tagging [51]. As shown in Figure 50, the transformation-based learning iteratively learns new rules as it operates on the corpus. The template rules used in [84] are shown in Figure 51.

We applied these two NLP techniques on several example pages and observed mixed results. As summarized in Figure 52, on some pages we saw good performance (marked G1, G2, G3). Conversely, in some other cases we found that the results of NP chunking cannot be relied upon directly–that there is a need for a hybrid approach (marked as H1, H2 in Figure 52).

- G1: In the result G1 in Figure 52, our original entity extractor returned "exchanged mosque" as one of the mosque names. As the chunking result shows, the two words, "exchanged" and "mosque," were separated into different chunks. Using this information, our entity extraction can be enhanced to prune out the mosque names that span words belonging to different chunks.

- G2: Likewise, consider the result of G2 in Figure 52. For this Web page, our entity extraction generated "fahd mosque" as a mosque name. The NP chunking result, however, considers "fahd mosque" as part of a noun phrase "wahhabi king fahd mosque." Our algorithm can use this noun phrase to generate the correct mosque name.

- G3: For the case of G3, our entity extraction produced two erroneous results: "topics red mosque" and "captured red mosque." As the result of the chunking shows, "topics" and "red mosque" are separated into



Word Patterns		Tag Patterns	
Pattern	Meaning	Pattern	Meaning
W_0	current word	T_0	current tag
W_{-1}	word 1 to left	T_{-1}, T_0	current tag and tag to left
W_1	word 1 to right	T_0, T_1	current tag and tag to right
W_{-1}, W_0	current word and word to left	T_{-2}, T_{-1}	two tags to left
W_0, W_1	current word and word to right	T_1, T_2	two tags to right
W_{-1}, W_1	word to left and word to right		
W_{-2}, W_{-1}	two words to left		
W_1, W_2	two words to right		
$W_{-1,-2,-3}$	word 1 or 2 or 3 to left		
$W_{1,2,3}$	word 1 or 2 or 3 to right		

Figure 51: Template of rules used in NP Chunking algorithm (image obtained from [84]).

different NP chunks. And for the second erroneous result, NP chunking places it as part of a larger noun phrase "the captured red mosque and militants," indicating that this is not the name of a mosque.

- H1: The case H1 shows an example page for which the results of NP chunking were not sufficient to fix the errors. Our entity extraction produced two erroneous mosque names on this page–"worshippers offered friday prayers mosque" and "bombed-out mosque." The first error could be fixed using NP chunking: it split out the words "worshippers" and "offered" from the noun phrase, "friday prayers mosque." However, for the second erroneous result "the bombed-out mosque" is indeed a noun phrase. Pruning out this later result would require other hybrid techniques besides noun phrase chunking.

- H2: The case of H2 also shows an example page for which noun phrase chunking results were not sufficient to correct the errors in our entity extraction. In particular, for this page, noun phrase chunking produced two candidates: "beautiful mosques" and "the blue mosque." The first phrase is not the name of the mosque, while the second phrase is a mosque name. We will need hybrid techniques here besides noun phrase chunking.

H	#mosque-name	NP Chunking Result
G1	exchanged mosque	locally/RB ,/, [we/PRP] joined/VBD with/IN [members/NNS] of/IN [new/JJ york/NN city/NN] s/VBZ [96/CD th/NN street/NN mosque/NN] for/IN [dialogue/NN] ,/, exchanged/VBD [mosque/NN and/CC synagogue/NN visits/NNS] and/CC worked/VBD side-by-side/RB in/IN [a/DT soup/NN kitchen/NN] run/VBN by/IN [a/DT local/JJ presbyterian/NN church/NN] ./.
G2	fahd mosque	[he/PRP] is/VBZ [imam/NN] at/IN [the/DT wal-mart-sized/JJ ,/, wahhabi/JJ king/NN fahd/NN mosque/NN] ,/, [which/WDT] opened/VBD in/IN [2000/CD] on/IN [the/DT outskirts/NNS] of/IN [sarajevo/NN] ,/, built/VBN with/IN [saudi/JJ money/NN] and/CC named/VBN after/IN [the/DT saudi/NN monarch/NN]
G3	topics red mosque, captured red mosque	[tech/JJ weather/NN topics/NNS] red/JJ mosque/NN [red/JJ mosque/NN] [109/CD usa/NN] [today/NN stories/NNS] about/IN [red/JJ mosque/NN prev/NN 4/CD 5/CD 6/CD 7/CD] [next/IN latest/JJS news/NN] from/IN [usa/NN] [today/NN world/NN associated/VBN press/NN writer/NN authorities/NNS] on/IN [Monday/NNP] probed/VBD [suspected/JJ links/NNS] between/IN [radicals/NNS] from/IN [the/DT captured/VBN red/JJ mosque/NN and/CC militants/NNS] in/IN [pakistan/NN] ['s/POS northwest/RB frontier/NN] ,/, where/WRB [73/CD people/NNS] died/VBD in/IN [weekend/NN suicide/NN attacks/NNS and/CC bombings/NNS] ./.
H1	worshippers offered friday prayers mosque, bombed-out mosque	[more/JJR than/IN 100/CD] injured/VBN in/IN [suicide/NN bombing/NN incident/NN] took/VBD [place/NN] as/IN [worshippers/NNS] offered/VBD [Friday/NNP prayers/NNS mosque/NN] frequented/VBN by/IN [officials/NNS] working/VBG at/IN [checkpoints/NNS] on/IN [nato/NN supply/NN routes/NNS]. [taliban/JJ and/CC al/JJ qaeda/NN militants/NNS] ./.Ûthe/DT bombed-out/JJ mosque/NN] had/VBD been/VBN frequented/VBN by/IN
H2	blue mosque	[cities/NNS] in/IN [afganistan/JJ including/VBG kabul/NN] ./.tudy/VB [the/DT afghanistan/JJ religion/NN] and/CC visit/VB [beautiful/JJ mosques/NNS] [the/DT blue/JJ mosque/NN] is/VBZ [beautifula/NN] as/RB well/RB [as/IN many/JJ other/JJ temples/NNS] throughout/IN [the/DT country/NN] ./.Ûthere/EX] are/VBP [many/JJ people/NNS] [that/WDT] are/VBP willing/JJ to/TO give/VB [you/PRP] [personal/JJ tours/NNS] including/VBG taking/VBG [you/PRP] outside/JJ of/IN [the/DT city/NN] to/TO experience/VB [native/JJ customs/NNS] ./.

Figure 52: Results of NP chunking on the example Web pages.

4 Discovery of Features from Geo-tagged Images

We are motivated to study how to benefit from the increasing prevalence of geo-tagged media on the open Web, for our problem of generating geospatial databases. We are observing that media content on the Web is increasingly becoming geo-tagged, *i.e.*, a geo-coordinate is being tagged with a piece of media content. Such geo-tagged media can be found in a variety of formats, *e.g.*, images on picture-sharing Web sites like *Flickr* and *Picasa*, videos on *YouTube*, rich editorial content on *Wikipedia*, and live status updates on real-time messaging services like *Twitter*.

4.1 Discovery of Mosque Features from *Flickr*

In the first portion of this task, we focused on studying how to use geo-tagged images in our problem of generating mosque features. We used images publicly available on *Flickr*, a popular photo sharing service. This service provides access to its content via its Application Programming Interface (API), with well maintained documentation available on its website at `http://www.flickr.com/services/api`.

Using the *Flickr* API, we obtained a total of 4246 "mosque-related" geo-tagged images all around the world, as visualized in Figure 53. We used the "flickr.photos.search" API. To restrict results to only those images that were geo-tagged, we used the search parameter "has_geo" and set its value to 1. Since our goal is to discover mosque features, we specified a list of mosque-related keywords, *i.e.*, {mosque, masjid, shrine}, in the "tags" criteria. We note that due to the preliminary nature of our investigation, the list of search terms is quite simplistic. In full deployment, a more extensive list of keywords, as well as searching within full text rather than just within the tags field, would be necessary.

We used the geo-coordinates of these mosque-related images to filter down to those relevant to our target geography of Afghanistan. We used a bounding box of latitude between 29 and 39, and longitude between 60 and 74, as a rough approximation of the boundary of Afghanistan. This filtering resulted in 45 images, as displayed in Figure 54. For each image, our database contained latitude and longitude as obtained from geo-tags, the URL of the image, and the title and description of the image.

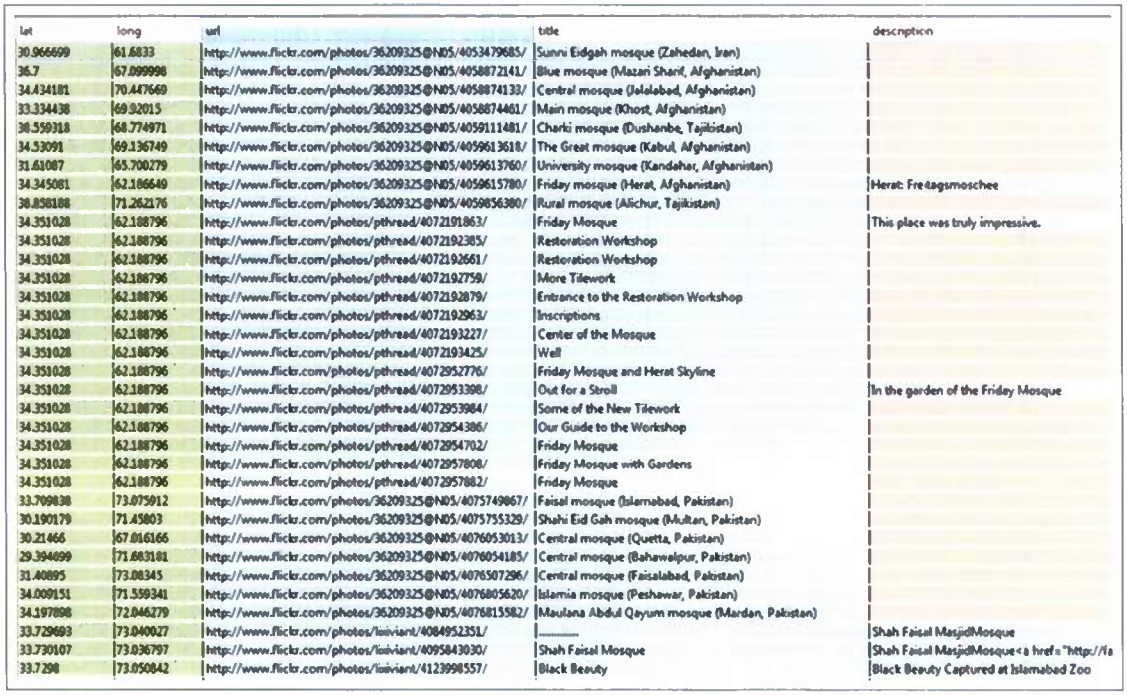
In this image dataset, we observed a consistent pattern in the title of the images. We observed that the title of these images often followed a common pattern, *i.e.*, "#mosque-name (#city, #country)." The images following such a pattern in the title were uploaded by different users, so this pattern seemed to be a naturally common way for people to title their images. For example, "Blue Mosque (Mazari Sharif, Afghanistan)" follows that common pattern. By matching the title of the image against this pattern, we can accurately extract the name of the mosque, as well as its location.

We used this insight in conjunction with our existing mosque name extraction module to extract the name of the mosques from these images. We first match the title of the images against the pattern of "#mosque-name (#city, #country)" by looking for the opening bracket "(," the closing bracket ")," and the comma. If the title matches this pattern, we use the corresponding terms to obtain the mosque name, and its city and country. For the images for which the title does not match this pattern, we apply our existing text extraction module to extract the name of the mosque from the title, as well as the description of the image.

As a result, we were able to obtain the mosque names for 31 out of the 45 images in our dataset. As shown in Figure 55, we added 3 additional columns to our database table–for storing the extracted mosque name, city name, and country name. We found that the title of 20 images successfully matched the pattern of "#mosque-name (#city, #country)," so for these 20 images, our extraction could find all three attributes–mosque name, city and country. For another 11 images, our text extraction module could find the name of the mosque from the text content of the image, *i.e.*, the title and the description of the images. For these 11 images, we could not find the name of the city or country. For the remainder of the 14 images (out of a total of 45 images) we could not find any of the 3 attributes.

The results of our preliminary study look quite promising. In particular, we observed the following two key possibilities:

Figure 53: Map overlay of the geo-tagged images related to mosques, obtained from *Flickr*.



lat	long	url	title	description
30.966699	61.6833	http://www.flickr.com/photos/36209325@N05/4053479685/	Sunni Eidgah mosque (Zahedan, Iran)	
36.7	67.099998	http://www.flickr.com/photos/36209325@N05/4058872141/	Blue mosque (Mazari Sharif, Afghanistan)	
34.434181	70.447669	http://www.flickr.com/photos/36209325@N05/4058874133/	Central mosque (Jalalabad, Afghanistan)	
33.334438	69.92015	http://www.flickr.com/photos/36209325@N05/4058874461/	Main mosque (Khost, Afghanistan)	
38.559318	68.774971	http://www.flickr.com/photos/36209325@N05/4059111481/	Charki mosque (Dushanbe, Tajikistan)	
34.53091	69.136749	http://www.flickr.com/photos/36209325@N05/4059613618/	The Great mosque (Kabul, Afghanistan)	
31.61087	65.700279	http://www.flickr.com/photos/36209325@N05/4059613760/	University mosque (Kandahar, Afghanistan)	
34.345081	62.186649	http://www.flickr.com/photos/36209325@N05/4059615780/	Friday mosque (Herat, Afghanistan)	Herat: Freitagsmoschee
38.858188	71.262176	http://www.flickr.com/photos/36209325@N05/4059856380/	Rural mosque (Alichur, Tajikistan)	
34.351028	62.188796	http://www.flickr.com/photos/pthread/4072191863/	Friday Mosque	This place was truly impressive.
34.351028	62.188796	http://www.flickr.com/photos/pthread/4072192385/	Restoration Workshop	
34.351028	62.188796	http://www.flickr.com/photos/pthread/4072192661/	Restoration Workshop	
34.351028	62.188796	http://www.flickr.com/photos/pthread/4072192759/	More Tilework	
34.351028	62.188796	http://www.flickr.com/photos/pthread/4072192879/	Entrance to the Restoration Workshop	
34.351028	62.188796	http://www.flickr.com/photos/pthread/4072192963/	Inscriptions	
34.351028	62.188796	http://www.flickr.com/photos/pthread/4072193227/	Center of the Mosque	
34.351028	62.188796	http://www.flickr.com/photos/pthread/4072193425/	Well	
34.351028	62.188796	http://www.flickr.com/photos/pthread/4072952776/	Friday Mosque and Herat Skyline	
34.351028	62.188796	http://www.flickr.com/photos/pthread/4072953398/	Out for a Stroll	In the garden of the Friday Mosque
34.351028	62.188796	http://www.flickr.com/photos/pthread/4072953984/	Some of the New Tilework	
34.351028	62.188796	http://www.flickr.com/photos/pthread/4072954386/	Our Guide to the Workshop	
34.351028	62.188796	http://www.flickr.com/photos/pthread/4072954702/	Friday Mosque	
34.351028	62.188796	http://www.flickr.com/photos/pthread/4072957808/	Friday Mosque with Gardens	
34.351028	62.188796	http://www.flickr.com/photos/pthread/4072957882/	Friday Mosque	
33.709038	73.075912	http://www.flickr.com/photos/36209325@N05/4075749867/	Faisal mosque (Islamabad, Pakistan)	
30.190179	71.45803	http://www.flickr.com/photos/36209325@N05/4075755329/	Shahi Eid Gah mosque (Multan, Pakistan)	
30.21466	67.016166	http://www.flickr.com/photos/36209325@N05/4076053013/	Central mosque (Quetta, Pakistan)	
29.394699	71.683181	http://www.flickr.com/photos/36209325@N05/4076054185/	Central mosque (Bahawalpur, Pakistan)	
31.40895	73.08345	http://www.flickr.com/photos/36209325@N05/4076507296/	Central mosque (Faisalabad, Pakistan)	
34.009151	71.559341	http://www.flickr.com/photos/36209325@N05/4076805620/	Islamia mosque (Peshawar, Pakistan)	
34.197898	72.046279	http://www.flickr.com/photos/36209325@N05/4076815582/	Maulana Abdul Qayum mosque (Mardan, Pakistan)	
33.729693	73.040027	http://www.flickr.com/photos/laaviant/4084952351/	[illegible]	Shah Faisal MasjidMosque
33.730107	73.036797	http://www.flickr.com/photos/laaviant/4095843030/	Shah Faisal Mosque	Shah Faisal MasjidMosque<a href="http://fa
33.7298	73.050842	http://www.flickr.com/photos/laaviant/4123998557/	Black Beauty	Black Beauty Captured at Islamabad Zoo

Figure 54: Dataset of the mosque-related geo-tagged images obtained from *Flickr* that are relevant to the geography of Afghanistan.

1. Inferencing via coordinate matching: We found that for many of the images, we could not extract the name of the mosque. However, these images were geo-tagged. Therefore, it is likely that multiple images, whose geo-coordinates are in close proximity, represent the same geo-spatial feature. Thus, while an image may not specify the name of the feature, we can group that image together with other images, based on matching their geo-coordinates. We can infer the mosque name of that image based on the mosque name extracted from those other images. In other words, it is sufficient for us to be able to extract the name of the feature, *i.e.*, mosque name, from at least a few of the images in a group.

2. Visualization over time axis: We noticed that the meta-data of these images also include the date when the picture was taken (or uploaded). Using multiple images of a specific geospatial feature over a number of years, we can visualize its evolution. Many of the geospatial features that are of interest to us represent buildings and other structures, and it will be useful to track their lineage over time.

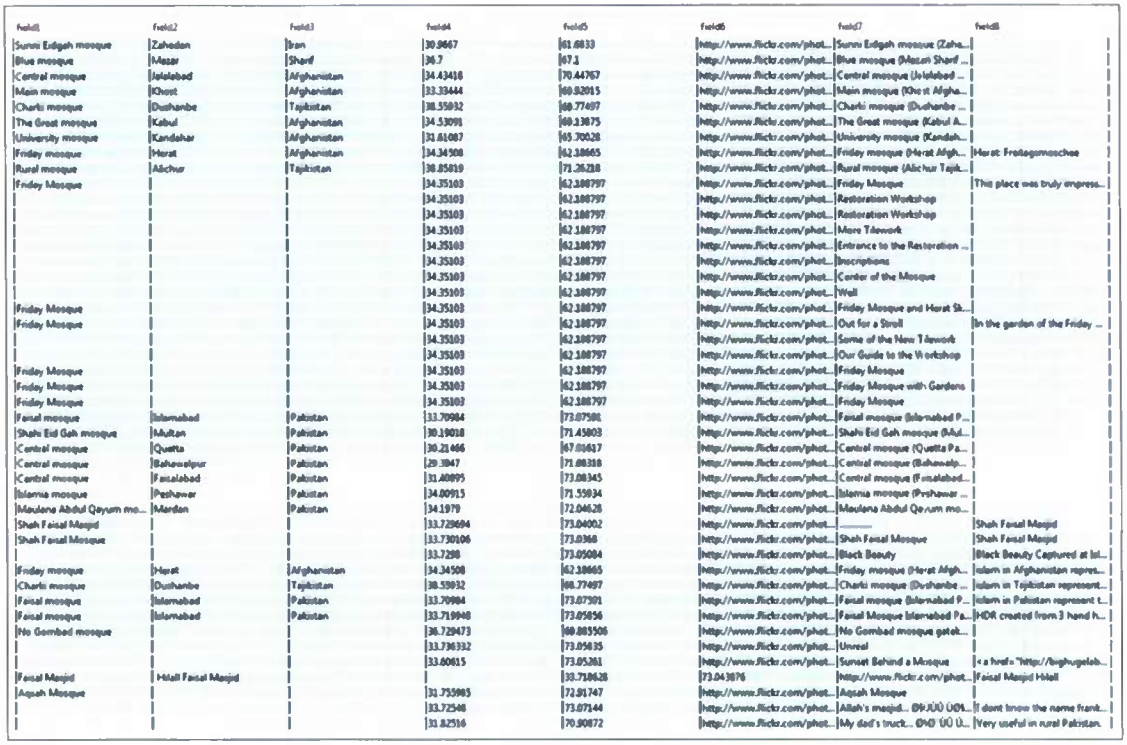
4.2 Creation of the *Media Aggregation Engine*

We have found that *Flickr* is an excellent source for obtaining Geo-tagged pictures. We have also discovered several more sites that allow API access to search for Geo-tagged media. Some of these sites include *Picasa* and *YouTube*. In order to facilitate our extraction, we have built a *Media Aggregate Engine*, a modular tool that allows us to pull and organize media from several sites. Currently it aggregates from *Flickr* [25], *Picasa* [17] and *YouTube* [8].

4.2.1 Discovery

We used the following search parameters within the API of our chosen sites:

- *Universal*:
  1. Tags: mosque, masjed, masjid, islamic center,
- *Flickr*:



field1	field2	field3	field4	field5	field6	field7	field8
Sunni Edgah mosque	Zahedan	Iran	30.9667	61.6833	http://www.flickr.com/phot...	Sunni Edgah mosque (Zaha...	
Blue mosque	Mazar	Sharif	36.7	67.1	http://www.flickr.com/phot...	Blue mosque (Mazar Sharif ...	
Central mosque	Jalalabad	Afghanistan	34.43416	70.44767	http://www.flickr.com/phot...	Central mosque (Jalalabad ...	
Main mosque	Khost	Afghanistan	33.33444	69.92015	http://www.flickr.com/phot...	Main mosque (Khost Afgha...	
Charki mosque	Dushanbe	Tajikistan	38.55932	68.77497	http://www.flickr.com/phot...	Charki mosque (Dushanbe ...	
The Great mosque	Kabul	Afghanistan	34.53091	69.13075	http://www.flickr.com/phot...	The Great mosque (Kabul A...	
University mosque	Kandahar	Afghanistan	31.61087	65.70028	http://www.flickr.com/phot...	University mosque (Kandah...	
Friday mosque	Herat	Afghanistan	34.34508	62.18665	http://www.flickr.com/phot...	Friday mosque (Herat Afgh...	Herat: Frotagemoschee
Rural mosque	Alichur	Tajikistan	38.85819	71.26238	http://www.flickr.com/phot...	Rural mosque (Alichur Tajik...	
Friday Mosque			34.35103	62.188797	http://www.flickr.com/phot...	Friday Mosque	This place was truly impress...
			34.35103	62.188797	http://www.flickr.com/phot...	Restoration Workshop	
			34.35103	62.188797	http://www.flickr.com/phot...	Restoration Workshop	
			34.35103	62.188797	http://www.flickr.com/phot...	More Tilework	
			34.35103	62.188797	http://www.flickr.com/phot...	Entrance to the Restoration ...	
			34.35103	62.188797	http://www.flickr.com/phot...	Inscriptions	
			34.35103	62.188797	http://www.flickr.com/phot...	Center of the Mosque	
			34.35103	62.188797	http://www.flickr.com/phot...	Well	
Friday Mosque			34.35103	62.188797	http://www.flickr.com/phot...	Friday Mosque and Herat Sk...	
Friday Mosque			34.35103	62.188797	http://www.flickr.com/phot...	Out for a Stroll	In the garden of the Friday ...
			34.35103	62.188797	http://www.flickr.com/phot...	Some of the New Tilework	
			34.35103	62.188797	http://www.flickr.com/phot...	Our Guide to the Workshop	
Friday Mosque			34.35103	62.188797	http://www.flickr.com/phot...	Friday Mosque	
Friday Mosque			34.35103	62.188797	http://www.flickr.com/phot...	Friday Mosque with Gardens	
Friday Mosque			34.35103	62.188797	http://www.flickr.com/phot...	Friday Mosque	
Faisal mosque	Islamabad	Pakistan	33.70984	73.07301	http://www.flickr.com/phot...	Faisal mosque (Islamabad P...	
Shahi Eid Gah mosque	Multan	Pakistan	30.19018	71.45803	http://www.flickr.com/phot...	Shahi Eid Gah mosque (Mul...	
Central mosque	Quetta	Pakistan	30.21466	67.01617	http://www.flickr.com/phot...	Central mosque (Quetta Pa...	
Central mosque	Bahawalpur	Pakistan	29.3947	71.68318	http://www.flickr.com/phot...	Central mosque (Bahawalp...	
Central mosque	Faisalabad	Pakistan	31.40095	73.08345	http://www.flickr.com/phot...	Central mosque (Faisalabad...	
Islamia mosque	Peshawar	Pakistan	34.00915	71.55934	http://www.flickr.com/phot...	Islamia mosque (Peshawar ...	
Maulana Abdul Qayum mo...	Mardan	Pakistan	34.1979	72.04628	http://www.flickr.com/phot...	Maulana Abdul Qayum mo...	
Shah Faisal Masjid			33.729694	73.04002	http://www.flickr.com/phot...	_____	Shah Faisal Masjid
Shah Faisal Mosque			33.730106	73.0368	http://www.flickr.com/phot...	Shah Faisal Mosque	Shah Faisal Masjid
			33.7298	73.05084	http://www.flickr.com/phot...	Black Beauty	Black Beauty Captured at Isl...
Friday mosque	Herat	Afghanistan	34.34508	62.18665	http://www.flickr.com/phot...	Friday mosque (Herat Afgh...	Islam in Afghanistan repres...
Charki mosque	Dushanbe	Tajikistan	38.55932	68.77497	http://www.flickr.com/phot...	Charki mosque (Dushanbe ...	Islam in Tajikistan represent...
Faisal mosque	Islamabad	Pakistan	33.70984	73.07301	http://www.flickr.com/phot...	Faisal mosque (Islamabad P...	Islam in Pakistan represent t...
Faisal mosque	Islamabad	Pakistan	33.71948	73.05856	http://www.flickr.com/phot...	Faisal Mosque Islamabad Pa...	HDR created from 3 hand h...
No Gombad mosque			36.729473	69.803506	http://www.flickr.com/phot...	No Gombad mosque gatek...	
			33.736332	73.05835	http://www.flickr.com/phot...	Unreal	
			33.60815	73.05261	http://www.flickr.com/phot...	Sunset Behind a Mosque	<a href="http://bighugelab...
Faisal Masjid	Hilali Faisal Masjid			33.730628	73.043676	http://www.flickr.com/phot...	Faisal Masjid Hilali
Aqsah Mosque			31.775985	72.91747	http://www.flickr.com/phot...	Aqsah Mosque	
			33.72548	73.07144	http://www.flickr.com/phot...	Allah's masjid... [illegible]	I dont know the name frank...
			31.82516	70.90872	http://www.flickr.com/phot...	My dad's truck... [illegible]	Very useful in rural Pakistan.

Figure 55: Mosque-related geo-tagged image dataset, after extraction of the entities–#mosque-name, #city, and #country.

1. Bounding Box (long, lat): (61.018066, 29.897806), (38.358888, 71.279297)
2. Accuracy: 12 (Between city and street level)
3. Has Geo: 1 (Return only pictures with associated geospatial information)

- *Picasa*:
  1. Bounding Box (long, lat): (61.018066, 29.897806), (38.358888, 71.279297)
- *YouTube*:
  1. Center Point (lat, long): (33.852170, 66.005859)
  2. Radius (miles): 200

Using these parameters, we discovered a total of 204 media items inside the rough bounding-box of Afghanistan. We postulated that some portion of these pieces of media might be tagged incorrectly or out of context *e.g.*, a picture of a drawing of a mosque instead of an actual mosque, and strove to isolate the correctly-tagged media items. A sample of context-correct and context-incorrect images can be seen in Figure 56.

4.2.2 Grouping

After reading several papers ([62, 87, 94]) we further postulated that it might be possible to eliminate mis-tagged media through geographic grouping, since it would be less likely for multiple pictures to be mis-tagged or tagged out of context. We used the following algorithm for our grouping:

- Choose a distance, Δ.
- Create a new group and place the first piece of media inside.

Context Correct
http://www.flickr.com/photos/78303790@N00/7454594
http://www.flickr.com/photos/90775848@N00/325093317
http://www.flickr.com/photos/7571511@N03/439873680
http://www.youtube.com/watch?v=4youfUPsecs
http://www.flickr.com/photos/22582876@N00/2043178240
Context Incorrect
http://www.flickr.com/photos/94482130@N00/863309218
http://www.flickr.com/photos/9692004@N07/729185829
http://www.youtube.com/watch?v=FyQuZ9QmJ_E
http://www.flickr.com/photos/78303790@N00/2538010998
http://www.youtube.com/watch?v=jmarVCZKPKo

Figure 56: Sample of context-correct and context-incorrect media items obtained from the *Media Aggregate Engine*.

- For the next piece of media, examine each existing group.
  - For each piece of existing media in the group, examine the distance between it and the currently held media.
  - If the distance is less that Δ for more than half of the group, place the currently held piece of media inside.
  - If not, examine the next group.
- If no groups are within Δ of the currently held piece of media, create a new group and place it inside.
- Repeat for all remaining ungrouped media.

Using this method and a Δ of .1 miles, we discovered that the pieces of media were divided into 24 groups, with 99 media items remaining ungrouped. We assigned an analyst to check the context of each of the media groups and 20 of the remaining ungrouped items. We found that 100% of the grouped media items remained context-relevant while only 75% of the ungrouped media items were context-relevant.

Based on this data, we discovered that it is true that grouped photos provide a better rate of accuracy. However, the ungrouped photos also contain a fairly high rate of accuracy (75%) and, even with the assumed incorrect quarter thrown out, generate more results than the the points given by photo groups (24 points in groups vs. 75% of 99, or approximately 74). It is, however, much more difficult to programmatically check the accuracy of each individual photo in the ungrouped section, making this much less useful for high-volume processing.

We also postulated that media groups containing contributions from only one user might act similarly to ungrouped media since their context is not corroborated by a second party. After analyzing several of the single-user media groups, though, we discovered that even these retain 100% context-correctness.

4.2.3 Grouping Radius

During our analysis of the affect of grouping media, we discovered that different grouping Δs created different distributions of groups and often differing numbers of groups overall. An example of this can be seen in figure 57.

We chose to move to a more media-rich area, Colorado, to continue our research on grouping radius. We used the following search parameters within our *Media Aggregate Engine*.

- *Universal*:
  1. Tags: mountain, peak, summit, ridge
- *Flickr*:
  1. Bounding Box (long, lat): (-109.0448, 37.004746), (-102.062988, 40.996484)
  2. Accuracy: 11 (Street level)

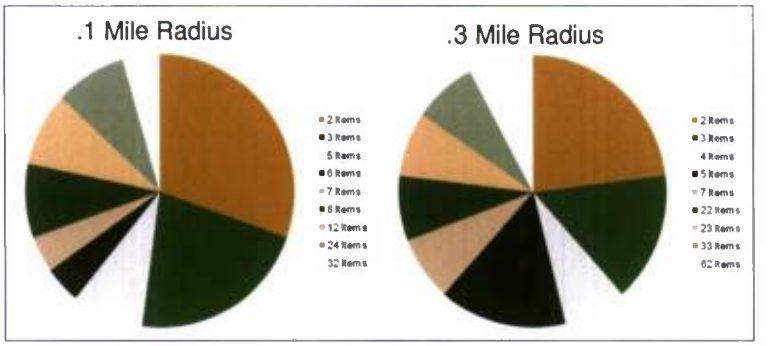


Figure 57: A distribution of mosque-related media group sizes across two grouping-radius Δs.

1 Mile	2 Miles	5 Miles
94	97	74

Figure 58: A distribution of mountain-related media group sizes across three grouping-radius Δs.

3. Has Geo: 1 (Return only pictures with associated geospatial information)

- *Picasa*:
  1. Bounding Box (long, lat): (-109.0448, 37.004746), (-102.062988, 40.996484)
- *YouTube*:
  1. Center Point (lat, long): (38.925229, -105.710449)
  2. Radius (miles): 150

Using these parameters and an upper-bound limit of the first 20 pages each from *Flicker* and *YouTube*, we discovered 5,476 media items. We further limited our observations to groups containing at least 10 media items. 4884 pieces of media fell into these groups. We chose to examine grouping-radius Δs of 1, 2 and 5 miles. Figure 58 briefly illustrates our findings.

We postulated that too large of a grouping radius will lead to many disparate groups being lumped together. Too small of a grouping radius, however, will lead to many repeat groups of the same feature in context. We also discovered that this is a very difficult task to quantify since it requires human analysis of large portions of data.

We chose to examine, in detail, the 1 mile radius groupings for our media gathered in Colorado. Figure 59 summarizes our findings from one such group. We discovered that many of the media items, even from disparate users, share similar tags. We also discovered that, and specifically in the group seen in figure 59, the groups contained only one distinct feature. This means that the grouping radius Δ of 1 mile for mountains is at least small enough to individualize the groups to disparate context items.

4.3 Grouping Benefits

We discovered two main benefits associated with grouping our media.

- Filtering of accuracy based on context of search
  - Works even if groups contain media items from only a single user.
- Media inferencing
  - The ability to apply shared traits to all media items in a group based on geo-location.

Media Item Location	Relevant Tags
http://www.flickr.com/photos/66063424@N00/4507303481	Longs Peak mountain
http://www.flickr.com/photos/7604710@N07/4270757322	Longs Peak mountain
http://www.flickr.com/photos/54565232@N00/3946896094	Chasm lake and Diamountain
http://www.flickr.com/photos/54565232@N00/3946896838	Mountain
http://www.flickr.com/photos/54565232@N00/3946894132	Glass lake, Chasm, longs peak
http://www.flickr.com/photos/54565232@N00/3946894738	Diamountain and chasm lake
http://www.flickr.com/photos/54565232@N00/3946088169	Lambs slide
http://www.flickr.com/photos/79808541@N00/2669800037	Chasm lake
http://www.flickr.com/photos/20693808@N05/2316526933	Longs Peak mountain
http://www.flickr.com/photos/20693808@N05/2316526725	Longs Peak mountain
http://www.flickr.com/photos/20693808@N05/2317334652	Longs Peak mountain
http://www.flickr.com/photos/20693808@N05/2316526245	Longs Peak mountain
http://www.flickr.com/photos/76209814@N00/612208266	Longs Peak mountain
http://picasaweb.google.com/oxelson/MeekerLongsJul2008#5229662569908698802	Longs Peak mountain
http://www.youtube.com/watch?v=2uSc_QJS5zw	Longs Peak mountain
http://www.youtube.com/watch?v=BkvaDTlA2uU	Longs Peak mountain
http://www.youtube.com/watch?v=pYYDILBqgDg	Longs Peak mountain
http://www.youtube.com/watch?v=QmPjov1pskM	Longs Peak mountain
http://www.youtube.com/watch?v=MvtxXki4w28	Longs Peak mountain
http://www.youtube.com/watch?v=60S_gr19stg	Longs Peak mountain

Figure 59: A sample group from the 1 mile radius grouped media.

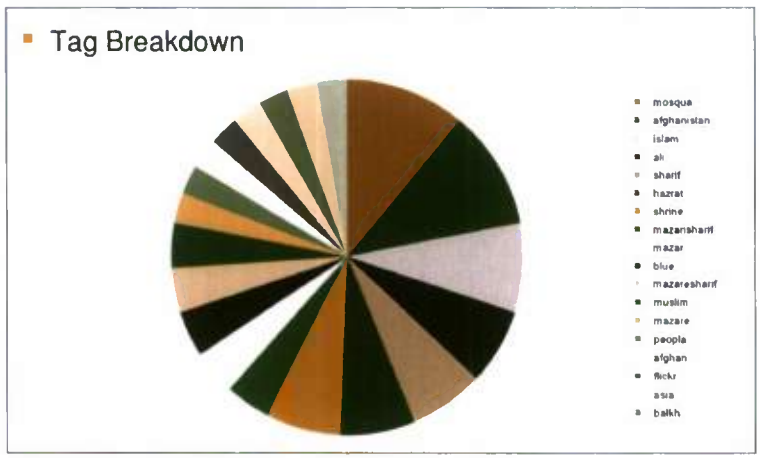


Figure 60: A distribution of tags within a single mosque-related media-group.

4.1: Filtering As we discovered in section 4.2.1, grouping provides 100% context-correct accuracy for our search term. This makes it easy to define certain locations provided by groups by the information gathered from within the group.

4.2: Inferencing We discovered that the type and frequency of tags within a single group can provide valuable information that we can use to apply traits across an entire group. Figure 60 shows a frequency distribution of tags across one particular mosque-related group. We see that general tags, *e.g.*, "mosque" and "afghanistan" appear very often, while more specific tags see a greatly reduced frequency. In figure 61, we have highlighted the primary tags of each media item in this group.

In analyzing and cross-checking tags through the URLs, we discovered that these media items are of a Mosque known as the Shrine of Hazrat Ali, also known as the Blue Mosque. It exists in the city Mazar-e Sharif. All

Media Item Location	Primary Tags
http://www.flickr.com/photos/28421453@N07/4044299773	Hazrat Ali, Mazar-i Sharif, Timurid
http://www.flickr.com/photos/28421453@N07/3931587585	Mazar-e, Sharif, hazrat, ali
http://www.flickr.com/photos/28421453@N07/3814131825	Hazrat, Ali, Mazar-e, Sharif, Shrine
http://www.flickr.com/photos/35604701@N07/3296151742	HAZRAT, ALI, MAZAR, SHARIF, BLUE, SHRINE
http://www.flickr.com/photos/35604701@N07/3295309181	HAZRAT, ALI, MAZAR, SHARIF, BLUE, SHRINE
http://www.flickr.com/photos/35604701@N07/3295291609	HAZRAT, ALI, MAZAR, SHARIF, BLUE, SHRINE
http://www.flickr.com/photos/35597531@N02/3295046993	Mazar-e-sharif, mazarisharif, hazara
http://www.flickr.com/photos/35597531@N02/3295045179	Mazar-e-sharif, mazarisharif, red

Figure 61: A sample group from mosque-related media with primary tags highlighted.

of this information can be found within the tags of the media. Some of the photos, #4 titled "Wrestling for Relics" and #5 titled "Afghan New Year" do not contain images of the Blue Mosque itself, rather it's grounds. By geo-coordinate association, though, and through their tags, we can add these sorts of descriptive attributes to photos in the group that do not already contain them. That is, by geo-coordinate association, if one piece of media contains information (such as an address or an alternate name) it is reasonable to assume that every photo should also contain this attribute.

5 Mountain Attribute Discovery

We discovered several existing sources for mountain feature information already on the internet. Among them were the *National Geospatial-Intelligence Agency* (hereafter *NGA*) [47], *PEAKLIST* [65], *Wikipedia* [66] and *GeoNames* [71], which uses the *NGA* database as its primary source. Of these, we found the *NGA* to be the most complete database, housing 14,028 unique mountain points within our area of interest, Afghanistan. The first attributes that we chose to add to this dataset were the relevant level 1 and level 2 Administrative Areas (hereafter ADMs), boundaries that are roughly analogous to States and Counties in the United States.

5.1 Administrative boundaries

5.1.1 ADM Boundary Dataset

After some initial research, we settled on the ADM information given by *GADM* [44]. From their website, we obtained a set of *Google Earth* KMZ files of the level 1 and level 2 ADM areas. From these we extracted individual

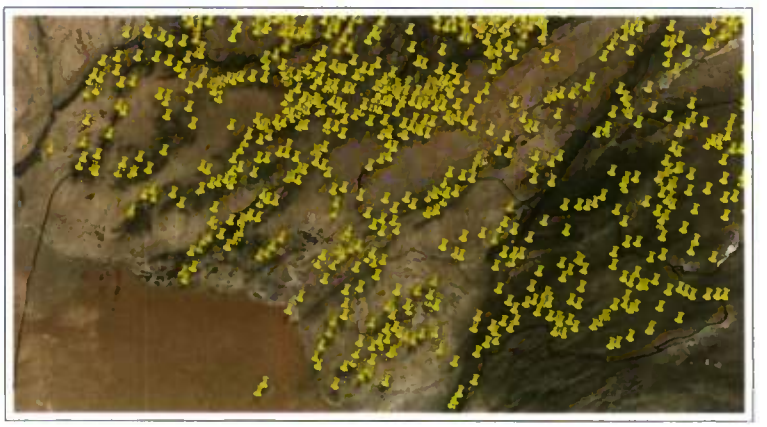


Figure 62: A close up of the province of Kandahar and surrounding area, each yellow push-pin represents a mountain feature.

Area	Recall	Precision (100 points)	Running Time
ADM 1	99.56% (13967)	100%	70 min 12 sec
ADM 2	98.26% (13789)	100%	136 min 41 sec

Figure 63: A summary of our findings for the original *Winding Algorithm*.

Area	Original Running Time	Optimized Running Time
ADM 1	70 min 12 sec	4 min 28 sec
ADM 2	136 min 41 sec	1 min 39 sec

Figure 64: A comparison of the original and optimized winding algorithms.

KML files which contain a description of the polygon bounding each ADM as a set of ordered points.

We were able to pull the sets of points into a *MySQL* database for further ease-of-processing. We repeated these steps to obtain the bounding polygons of all ADM level 1 and level 2 areas within Afghanistan. Finally, we plotted the location of each mountain peak on top of the ADM map, as seen in Figure 62. As this figure shows, mountains exist within many provinces in Afghanistan.

5.2 Boundary Matching

5.2.1 First Placement Method: Winding Algorithm

1: Original Winding Algorithm

For our first analysis, we chose to use a point-in-polygon algorithm commonly known as the *Winding Algorithm*. This algorithm calculates the number of times an infinite ray passes from the point to be tested across a set of directed vectors given by the points that make up the polygon's boundary. Counting around the polygon clockwise, vectors that cross the ray from below to above assign a winding number of $+1$ while vectors that cross the ray from above to below assign a winding number of -1. The total winding number is found by taking the sum of all the vector's winding numbers. A non-zero winding number means that the point is within the polygon.[3]

A table of our results can be seen in Figure 63. We assigned an analyst to examine our points and found a very high rate of accuracy; 100% for the 100 points that were checked against our plotted *Google Earth* KMZ file of both ADM level 1 and level 2. We found, though, that the algorithm as-is takes an interminably long time to run and wondered if there might be a method for optimization.

2: Optimized Winding Algorithm

We discovered that we can take the following steps to optimize our winding algorithm:

- For each polygon, store a bounding box given by the bottom-left and top-right most corners.
- For a given point, first check against the bounding box.
  - If the point falls within the box, it might be roughly within the given boundary, so run the winding algorithm on it.
  - Otherwise, skip this boundary

We found that this produced exactly the same level of recall and accuracy as our original algorithm while dramatically decreasing its running time. Figure 64 shows a comparison between the two winding methods.

5.2.2 Second Placement Method: Bounding Algorithm

1: First Pass Bounding Algorithm

We wondered if there might be another method to determine if a mountain point exists inside of a particular ADM polygon and developed our own *Bounding Algorithm* to test this hypothesis. Our algorithm operates as follows:

[3]See *this site* [90] for a more complete overview of point-in-polygon algorithms and the Winding Number.

Area	Recall	Precision
ADM 1	93.75% (13152)	100%
ADM 2	86.22% (12096)	100%

Figure 65: A summary of our findings from the *Bounding Algorithm*.

- Get latitude, ϕ, and longitude, λ, for a given mountain in decimal format.
- Choose a deviation, Δ, as a range for discovering boundaries.
- Determine all boundaries North of the mountain point.
  - Query the SQL table for all boundaries where:
    $\lambda_{(boundary)} \geq \lambda_{(mountain)}$ *and* $\phi_{(boundary)} - \Delta < \phi_{(mountain)} < \phi_{(boundary)} + \Delta$
    That is, where the longitude of the boundary point is greater than the mountain point's longitude and the latitude of the boundary point is within the deviation of the latitude of the mountain point.
  - Store all boundaries in an array.
- Determine all returned boundaries South of the mountain point.
  - Query the SQL table for all boundaries where:
    $\lambda_{(boundary)} \leq \lambda_{(mountain)}$ *and* $\phi_{(boundary)} - \Delta < \phi_{(mountain)} < \phi_{(boundary)} + \Delta$
    That is, where the longitude of the boundary point is less than the mountain point's longitude and the latitude of the boundary point is within the deviation of the latitude of the mountain point.
  - Store all returned boundaries in an array.
- Determine all boundaries East of the mountain point.
  - Query the SQL table for all boundaries where:
    $\phi_{(boundary)} \geq \phi_{(mountain)}$ *and* $\lambda_{(boundary)} - \Delta < \lambda_{(mountain)} < \lambda_{(boundary)} + \Delta$
    That is, where the latitude of the boundary point is greater than the mountain point's latitude and the longitude of the boundary point is within the deviation of the longitude of the mountain point.
  - Store all returned boundaries in an array.
- Determine all boundaries West of the mountain point.
  - Query the SQL table for all boundaries where:
    $\phi_{(boundary)} \leq \phi_{(mountain)}$ *and* $\lambda_{(boundary)} - \Delta < \lambda_{(mountain)} < \lambda_{(boundary)} + \Delta$
    That is, where the latitude of the boundary point is less than the mountain point's latitude and the longitude of the boundary point is within the deviation of the longitude of the mountain point.
  - Store all returned boundaries in an array.
- *Find* $\{North\} \cap \{South\} \cap \{East\} \cap \{West\}$
- If only one name exists in the intersection, it must be the correct ADM of the mountain point.
- Repeat for all $\Delta \in \{1, .1, .01, .001\}$

We discovered that large values of Δ almost always provide at least one match for our ADM boundary, but often provide many more than that. Figure 66 shows the number of ADM boundary matches as our Δ decreases for ADM level 1 and Figure 67 shows the same relationship for ADM level 2.

The results for our bounding algorithm are summarized in Figure 65. We assigned an analyst to determine the accuracy of our algorithm and found it to be 100% after checking 100 points against our plotted *Google Earth* KMZ file of both ADM level 1 and level 2. We found that our recall was slightly lower than that of the *winding algorithm* for ADM 1 and significantly lower than that of the *winding algorithm* for ADM 2.

ADM 1	5	4	3	2	1	0
Δ 1	12	135	1838	7558	4485	0
Δ .1	0	0	25	2027	11951	25
Δ .01	0	0	1	143	7370	6514
Δ .001	0	0	0	2	45	13981

Figure 66: A comparison of the number of ADM 1 boundary matches against reducing our Δ precision.

ADM 2	5	4	3	2	1	0
Δ 1	22	322	2422	6520	4709	33
Δ .1	0	2	180	3941	9789	116
Δ .01	0	0	1	153	6785	7089
Δ .001	0	0	0	1	20	14007

Figure 67: A comparison of the number of ADM 2 boundary matches against reducing our Δ precision.

2: Second Pass Bounding Algorithm

We modified our original *bounding algorithm* slightly in order to catch the mountain points that the first pass missed. Our *probabilistic mapping algorithm* is very similar to the *bounding algorithm* except that instead of iterating over a reducing Δ to find a boundary that provides a single match, we aggregate all possible boundaries in the North, South, East and West arrays. We then take $\{North\} \cup \{South\} \cup \{East\} \cup \{West\}$ and assign the most popular boundary as the mountain point's ADM. As the "Trust" score of our decisions, we used the ratio of the frequency of the most popular ADM feature in the combined array to that of the second most popular ADM feature.

The results for our probabilistic mapping algorithm are summarized in Figure 68. We assigned an analyst to examine several of the ADM boundaries for both level 1 and 2 and found that we had achieved roughly 55% precision with the remainder of our points.

5.2.3 *Winding Algorithm* and *Bounding Algorithm* comparison

We chose to directly compare the *Winding Algorithm* with the first pass results from our *Bounding Algorithm*. A comparison of running times can be seen in Figure 69. A comparison of plotting differences can be seen in Figure 70.

We see a very high correlation between the two algorithm approaches:

- 99.76% and 99.62% match for first pass Bounding vs Winding on ADM 1 and 2, respectively.
- 97.67% and 95.95% match for the second pass Bounding vs Winding on ADM 1 and 2, respectively.

Figure 71 shows a map of the 33 plotted points that differ between the *Bounding Algorithm* and the *Winding Algorithm*. We also see that the running times of the *Optimized Winding Algorithm* and the *Bounding Algorithm* are similar, though the Winding Algorithm is clearly faster. It may be possible to optimze the Bounding Algorithm in order to reach much closer speeds.

5.3 Mountain Elevation

5.3.1 GeoNames Elevation Discovery

We discovered that *GeoNames* has a webservice [70] that allows us to send a set of coordinates in decimal degrees and returns an elevation in meters. We stored these return values in a MySQL database for further analysis. We determined that this method had a recall of 96.84%, with a dummy value of -32768 being returned when *GeoNames* had no data. Figure 73 shows a binning analysis of its accuracy.

Area	Recall	Precision
ADM 1	100% (876)	50.74% (270 points)
ADM 2	100% (1932)	60.60% (264 points)

Figure 68: A summary of our findings from the *Probabilistic Mapping Algorithm.*

Area	Bounding	Winding	Optimized Winding
ADM 1	10 min 15 sec	70 min 12 sec	4 min 28 sec
ADM 2	21 min 30 sec	136 min 41 sec	1 min 39 sec

Figure 69: A comparison of the running times of the *Bounding Algorithm* with those of the *Winding Algorithm.*

5.3.2 SRTM CSI

In order to provide 100% recall and also have another set of elevations to compare, we chose to interpolate our own peak data over the SRTM CSI dataset. We downloaded the dataset from http://srtm.csi.cgiar.org/ [68] and extracted the elevation information from the files around Afghanistan. Since the SRTM CSI data provides elevation at 90m resolution, we decided that we must implement our own form of interpolation to discover peak heights that fall within this resolution.

1: Interpolation Method

We chose to use an inverse distance weighting interpolation method known as Shepard's Method. [67] Shepard's Method consists of the following sum:

$$u(x) = \sum_{k=0}^{N} \frac{w_k(x)}{\sum_{k=0}^{N} w_k(x)} u_k \tag{2}$$

where

$$w_k(x) = \frac{1}{d(x_1, x_2)^p} \tag{3}$$

Using this method, $u(x)$ will yield our interpolated elevation, where u_k is the elevation of each known SRTM point in our set. We control two factors, the subset of SRTM points, N and the "power parameter", p. N is manipulated through a set of distance, Δ, that we choose around our unknown mountain point. A higher Δ yields a higher N and thus a larger set to interpolate around. The "power parameter" determines the smoothing of our interpolated figures. When $0 < p < 1$, we expect a smoother interpolation, since distance away from our mountain point is not very heavily weighted. When $p > 1$, distance away from our mountain point is heavily weighted and our interpolation is expected to peak much more sharply. Despite these two variable parameters, experimentation shows that variance in either N or p does not significantly change our interpolated elevation. See Figures 74 and 75 for a comparison across the nine mountain points from Figure 72 .

2: Interpolation Results

Since we interpolated our specific mountain points over many points, we were able to achieve a recall of 100%. Figure 76 shows a binning analysis of its accuracy.

5.3.3 Conclusion

We note that when elevation information exists from the *NGA Gazetteer*, both *GeoNames* and *Shepard's Method* fall close to that than the elevation taken from specific Internet sources. In either case, sometimes *GeoNames* and *Shepard's Method* fall close to the mountain point's actual elevation, *e.g.*, Shāh Fūlādī, while sometimes it falls far apart, *e.g.*, Kūh-e Fergardī.

We also notice one aberration, the *NGA Gazetteer*'s listed elevation for Shāh Ţūs Āqā Ghar is roughly three times the values given by the *Internet*, *GeoNames* and *Shepard's Method.* In fact, we notice that there are 64 mountain points whose *NGA* elevations are > 8000 while their interpolated results are much less. We notice

Area	First Pass	Second Pass
ADM 1	33	326
ADM 2	53	567

Figure 70: A comparison of the plotting differences between the *Bounding Algorithm* and the *Winding Algorithm*.

Figure 71: A map showing the plotting differences between the *Bounding Algorithm* and the *Winding Algorithm*. for ADM level 1.

further that the ratio of the *NGA* elevation over interpolated results is roughly 3. From this we postulate that these mountain points might have elevations incorrectly entered in feet rather than meters.

We discovered that between *GeoNames* and *Shepard's Method*, and as illustrated in Figure 77, that *GeoNames* creates a better "guess" for a mountain's height, generally falling within approximately 100 meters. We postulate that they must use a different interpolation method than *Shepard's* and that it might be possible to increase our accuracy by experimenting with different interpolation methods.

We also discovered that interpolation has the secondary utility of verifying existing data, as we've seen in the 64 mountain point elevations that were probably entered in feet rather than meters.

Mountain	*NGA*	*Internet*
Kūh-e Jang Qal'eh	3781m	4171m
Gora Takurgar	None	3191m
Shāh Ţūs Āqā Ghar	15,758m	4803m
Shāh Fūlādī	None	4153m
Kūh-e Pīshashgal	4693m	6290m
Kūh-e Bandaka	6271m	6812m
Kūh-e Fergardī	None	5096m
Noshaq	7482m	7492m
Koh-i-Safēd Khe s	None	5325m

Figure 72: A comparison of elevations from the *NGA* database with information taken from *PeakList* and *Wikipedia*.

Difference in Meters	Number of Occurrences
0	33
< 10	663
$10 - 100$	3706
$100 - 1000$	415
1000+	88

Figure 73: A distribution of elevation differences between the existing *NGA* Gazetteer and values returned from *GeoNames*.

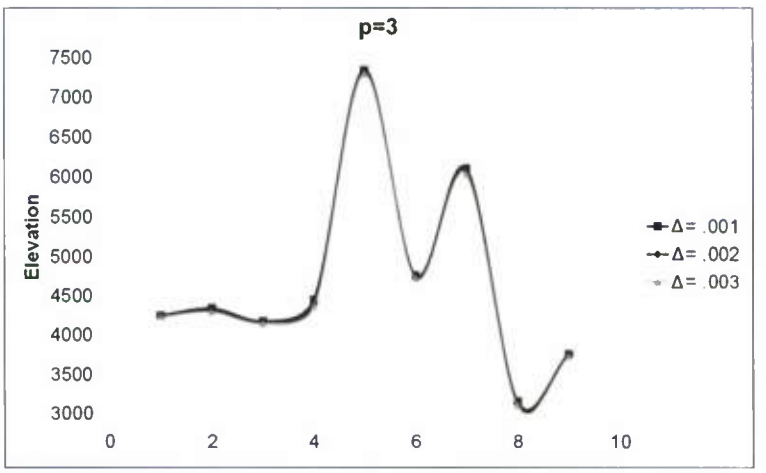


Figure 74: A comparison of changing $\Delta \in \{.001, .002, .003\}$ with $p = 3$ on interpolated elevation.

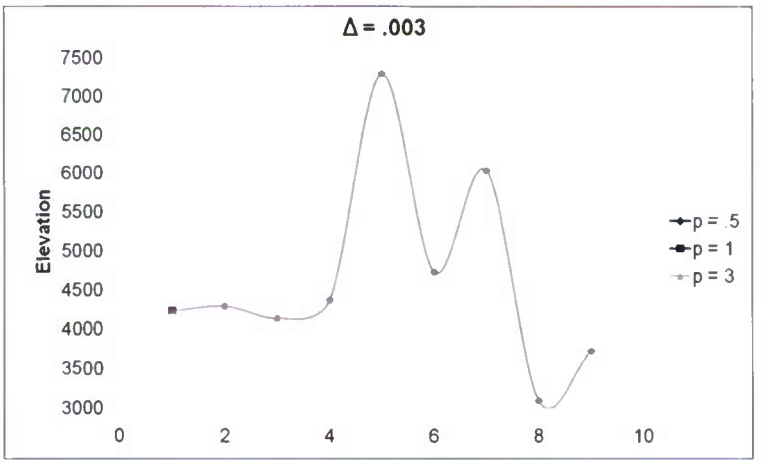


Figure 75: A comparison of changing $p \in \{.5, 1, 3\}$ with $\delta = 0.003$ decimal degrees on interpolated elevation.

Difference in Meters	Number of Occurrences
0	2
< 10	43
$10 - 100$	2665
$100 - 1000$	2197
1000+	95

Figure 76: A distribution of elevation differences between the existing *NGA* Gazetteer and values returned from our *Shepard's Method* interpolation.

Mountain	*NGA*	*Internet*	GeoNames	Shepard's Method
Kūh-e Jang Qal'eh	3781m	4171m	3766m	3727m
Gora Takurgar	None	3191m	3168m	3097m
Shāh Ţūs Āqā Ghar	15,758m	4803m	4753m	4726m
Shāh Fūlādī	None	4153m	4153m	4138m
Kūh-e Pīshashgal	4693m	6290m	4352m	4294m
Kūh-e Bandaka	6271m	6812m	5814m	6037m
Kūh-e Fergardī	None	5096m	4220m	4232m
Noshaq	7482m	7492m	7172m	7294m
Koh-i-Safēd Khe s	None	5325m	4478m	4368m

Figure 77: A comparison of elevations from the *NGA* database with information taken from *PeakList* and *Wikipedia* with *GeoNames* and *Shepard's Method* included.

6 Neighborhood Extraction

While examining the categorization of mountain points within ADM level 1 and 2 areas, we found one major problem with the boundaries of such areas: namely that, depending on the country in question, the borders may be only approximately mapped and subject to frequent change.

This same uncertainty applies to neighborhood boundaries within a city. Neighborhoods are described by urban scholar Lewis Mumford in the following way: "Neighborhoods, in some primitive, inchoate fashion exist wherever human beings congregate, in permanent family dwellings; and many of the functions of the city tend to be distributed naturally—that is, without any theoretical preoccupation or political direction–into neighborhoods."[81] Since neighborhoods are defined by their residents instead of by a central political body, the boundaries between neighborhoods can fluctuate wildly, even as according to two of a given area's residents.

There are already several well tested methods for defining neighborhood boundaries. In the 1970s, researchers in Chicago, IL went door to door, asking residents to define the neighborhood they lived in. Given the residents answer and their address, the researchers were able to map out a rough set of boundaries for Chicago's over 200 neighborhoods. More recently we saw an application of the data from *Flickr*, a image sharing website, for discovery of neighborhood boundaries [4]. The images that users share via *Flickr* are often geo-tagged (using either coordinates derived from GPS-enabled digital cameras or those entered manually by users); in addition, users also add relevant keyword tags to their images. Using these two pieces of information, it was possible to determine the boundaries of colloquial features, such as neighborhoods.

Since these methods have been proven, we chose to find another way to quickly and programmatically define neighborhood boundaries for a given city. We chose to examine the data associated with the set of apartment and restaurant (hereafter: "feature") listings for a given city. That is, we examine the feature's title, description and any other associated meta-data in an attempt to extract its the neighborhood name within which it belongs. If a neighborhood name is located, we extract the feature's address and store it in a database. Using this database, we are able to predict approximate boundaries for each neighborhood within the city.

6.1 Area of study

We chose Chicago, IL as our area of study for the neighborhood task. Chicago is a suitable city for three major reasons: it is populous, information-rich and has a fairly complete set of boundary points already defined.

We prefer a populous area for our first study for several reasons. A populous city is much more likely to contain a large and diverse set of neighborhoods, whereas a less populated city might contain only a few neighborhoods or neighborhoods largely defined by large contract housing developers instead of being grown in an organic fashion by communities. A populous city is also more likely to contain a large number of apartment and restaurant features from which we might extract neighborhood boundaries. This becomes very important, since our extraction is unlikely to define exact areas. We rely on a large, statistically significant group of feature points to define our boundaries.

In saying that we require a city to be information-rich, we mean that a large percentage of its features have been harvested, indexed and made available to the public in a structured form on the Internet. We know that Chicago, IL is information-rich because a casual *Google* search of "Restaurants in Chicago, IL" will yield thousands of results. A similar search on a large, but information-sparse city such as Baghdad, Iraq, whose population is over 6,000,000, only yields nine results. Figures 79 and 80 contrast the results between Chicago and Baghdad.

In order to test the accuracy of our boundary definition methods, we wanted to choose a city whose boundaries were already approximately mapped out. Chicago has had several such neighborhood boundary surveys and keeps its information up to date using Geographic Information Systems technology. We located a project that created a *Google Earth* KMZ file of 227 of Chicago's neighborhoods [3]. Using this file, in a manner similar to our extraction of ADM level 1 and 2 boundaries in Afghanistan, we were able to obtain a set of points describing each neighborhood and store them in a database for later comparison. See figure 78 for a visual representation of the KMZ map.

6.2 Sources

6.2.1 Apartment Features

We chose to use apartment features as part of our test data because, given a highly populous and information-rich area, there is likely to be a large set of feature data publicly available for extraction and examination.

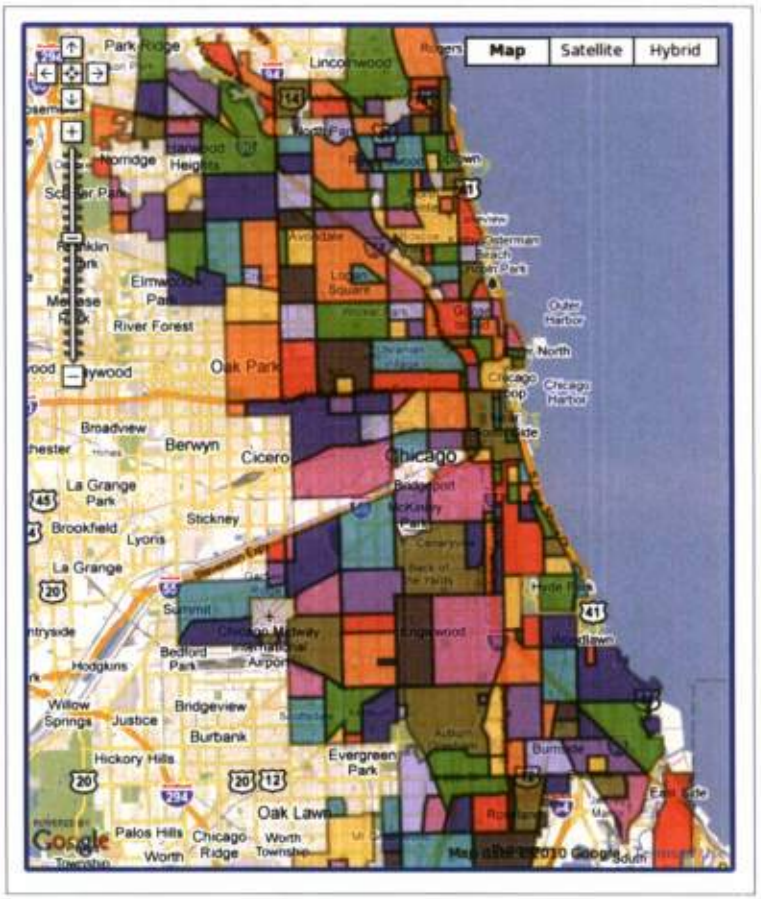


Figure 78: A color coded map of the *Google Earth* KMZ file which was used to parse neighborhood boundaries.

1: Apartment Features Source

There are many third-party apartment listings sites on the Internet. A short survey exposes three of the most popular: `www.apartments.com` [22], `www.mynewplace.com` [34] and `www.move.com` [32]. An examination of each of these sites show that they contain 490, 417 and 86 listings respectively; considering our need for large feature sets, we found these datasets to be too small. We postulate that their size is due to the sites' methods of collection: namely that they request feed-style submissions from apartment realtors; that is, their listing must be discovered and compiled manually.

Rather than crawl many of these sites manually to acquire our dataset, we turned to our existing apartment search technology, available at `apartments.cazoodle.com` [2]. Using our existing apartment listing aggregation methods, we were able to obtain a total of 36,320 apartment features for the Chicago, IL area.

1.1: Neighborhood Name Extraction

Our apartment feature listings do not have an explicit neighborhood field associated with them, so we could not categorize them exactly. Instead, we chose to examine each apartment feature's associated description in an attempt to classify it as a particular neighborhood. We used two methods of classification, multiple matching and single matching.

1.1.1: Multiple Matching

For our multiple matching algorithm, we chose to store an individual entry for each instance where a neighborhood name existed within the apartment feature's description. That is, given an entry such as: "Just walk the area and find everything you need, just a couple blocks away! So close to Wicker Park, Bucktown, and just a quick bus ride to Lincoln Park and Lakeview. Blue line can get you downtown in 20 min. This is the giant 3BR

Figure 79: A distribution of restaurants across Chicago, IL as shown by *Google Maps*. Each red dot represents a restaurant feature.

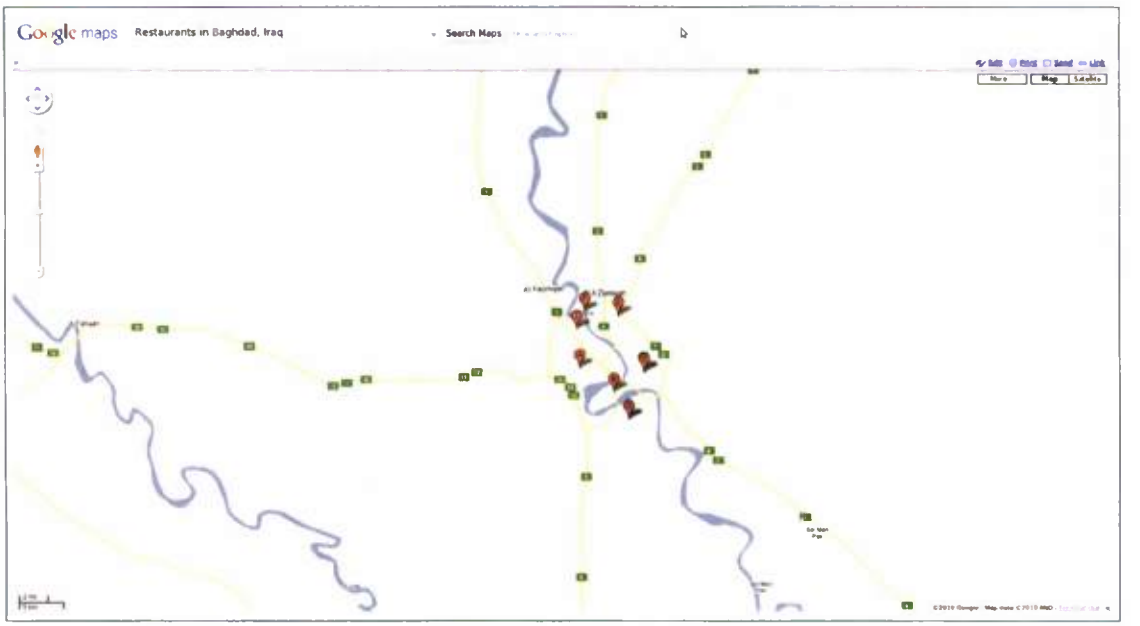


Figure 80: A distribution of restaurants across Baghdad, Iraq as shown by *Google Maps*. Each red dot represents a restaurant feature.

w/ huge kitchen and equal sized rooms that you've been looking for!!! New bathroom. Brand new kitchen w/ dishwasher and all new cabinets and tile. Plenty of sunny windows. Tons of living/dining area space. Large Back Porch," we would store an entry for "Wicker Park," "Bucktown," "Lincoln Park," and "Lakeview." We chose to do this because it becomes very difficult to programmatically determine which of the associated neighborhoods the apartment feature exists inside instead of simply nearby. Additionally, since we're not attempting to determine

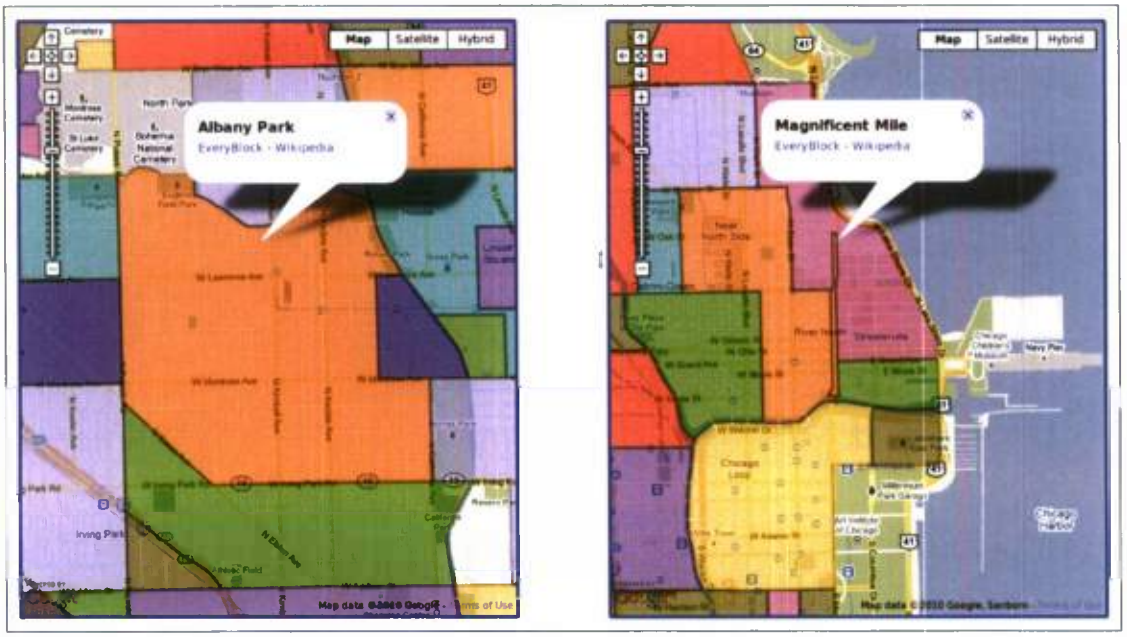


Figure 81: Two of Chicago, IL's neighboroods: Albany Park (left) and Magnificent Mile (right), to scale.

exact boundaries, it may be helpful to allow the entire set of associated neighborhoods to occupy the point, if the neighborhood boundaries are, in fact, nearby. Using this method we were able to extract 13,879 apartment features, each matched with an associated neighborhood.

1.1.2: Single Matching

We postulated that it might be possible to increase the accuracy of our extraction if we focused on apartment features whose description matched only one neighborhood. Using this method, we ignored records similar to the Multiple Matching example and only stored apartment features whose description matched exactly one neighborhood. Using this method we were able to extract 8,854 apartment features, each matched with an associated neighborhood.

1.2: Geo-Coding Addresses

Since the set of neighborhood boundary points we extracted from the *Google Earth* KMZ file are geo-coded latitude/longitude points, the points representing each of our apartment features must be in the same format for analysis. An advantage of using our own apartment dataset is that *Cazoodle* already has a process in place to convert each listing's street address into the corresponding latitude/longitude point, so no additional work needed to be done for our apartment feature comparison.

1.3: Accuracy Analysis

At this stage in our investigation, we were only interested in a basic examination of ground truth accuracy. We wanted to determine, for each apartment feature's point, if the point existed within the correct neighborhood. This obviously ignores points that might lie just outside a neighborhood boundary, but might still be valid since the boundary is only socially defined and may vary.

To calculate our accuracy, we used the same point-in-polygon *winding algorithm* as was used to determine our accuracy in mountain-point matching against ADM areas. This algorithm calculates the number of times an infinite ray passes from the point to be tested across a set of directed vectors given by the points that make up the polygon's boundary. Counting around the polygon clockwise, vectors that cross the ray from below to above assign a winding number of $+1$ while vectors that cross the ray from above to below assign a winding number of -1. The total winding number is found by taking the sum of all the vector's winding numbers. A non-zero winding number means that the point is within the polygon. A more detailed description can be found at `http://softsurfer.com/Archive/algorithm_0103/algorithm_0103.htm#WindingNumber`.

We ran the winding algorithm against each of our recorded apartment features latitude/longitude points, classifying each within the neighborhood boundary that the point actually lies in, according to our set of extracted boundary points. Of the 13,879 multiple match and 8,854 single match points, we were able to classify 11,233 and 7,277 points within neighborhoods, respectively. We believe that the winding algorithm was unable to classify 100% of the feature points because we believe that a small subset of the points gathered by our matching algorithms fall outside of all extracted neighborhood boundaries.

We found that 27.08% of points from our multiple matching and 32.55% of our single matching apartment feature points agreed with the winding algorithm completely. We found that, in general, the larger areas had high match percentages while the smaller neighborhood areas had very small match percentages. The Chicago neighborhood *Magnificent Mile*, for example, had 401 apartment features from the multiple matching algorithm, but only 1.99% actually fell within its area; the *Magnificent Mile* is a very small area of land near *The Loop*, on Chicago's eastern coast side. *Albany Park*, however, which is many times larger than the *Magnificent Mile*, contains 99 apartment feature points, with an accuracy rating of 61.61%. See figure 81 for a size comparison between *Albany Park* and *Magnificent Mile*.

We found to be true that our single matching algorithm returned a higher percentage of exact matches, as expected. We also realize that exact matching is only a basic analysis of our accuracy, but at this stage we believe it is an indicator that our apartment feature matching is reasonably accurate.

6.2.2 Restaurant Features

Similar to apartment features, we chose to include restaurant features in our examination because, for a highly populous and information-dense area, the set of features available for extraction should be sufficiently large for our analysis.

2: Restaurant Features Sources

Unlike our apartment features, *Cazoodle* does not have an existing dataset for immediate analysis. We chose to examine three primary sources for restaurant feature extraction, *Google*, *Yahoo* and *Boorah*.

2.1.1: Restaurant Extraction using *Google*

Ideally we hoped to use *Google*'s *local search* API to access restaurant data for Chicago, IL [6]. The *local search* API allows for the user to specify a query and a rough area. The API will then return a set of features that match the query and are local to the area specified. Results were initially promising; *Google* promised roughly 15,000 results for our query, "Restaurants in Chicago, IL." Unfortunately limitations within *Google*'s API allowed us to only return the first 64 results before refusing additional access. Rather than a mass information-retrieval service, the API seems to be intended for use within an existing website, to show a small subset of local results for a particular user query. Since the API limitations only allowed us to retrieve a small subset of the required data, we chose not to use *Google* for our final analysis.

2.1.2: Restaurant Extraction using *Yahoo*

Yahoo hosts a *local search* API very similar to *Google*'s; it allows the user to specify a query and a location, though it also allows the user to specify a much tighter radius for its area bound [9]. Similar to *Google*, also, *Yahoo*'s API seems focused more on providing information in a small sidebar to an existing website rather than for mass data extraction.

Rather than attempt to obtain data through *Yahoo*'s public API, we chose to extract directly from their search results page using our existing *Agent Building* technology. In this way, we were able to collect 8,853 total results.

2.1.3: Restaurant Extraction using *Boorah*

www.boorah.com is a web service that allows users to submit and rate restaurants in a given location [23]. Users are allowed to either "boo" (disapprove of) or "rah" (approve of) a given restaurant. The site tracks reviews, descriptions and various categories of rating. It hosts a list of 7,354 restaurant features within the Chicago, IL area.

We again leveraged our existing *Agent Building* technology in order to extract records from *Boorah*'s web service. During extraction, we realized that *Boorah* was limiting us to 1,000 results per category. In order to attempt to retrieve all 7,345 results, we chose to narrow each search by a given food category; that is, though 7,345 results appear in their listing, only 140 of those exist within the "seafood" category. Unfortunately their categories are ill-defined, making it difficult to programmatically extract all of their results. Using this piecemeal extraction method, we were able to obtain 3,560 restaurant features from their website.

2.2: Additional Sources

Additional sources for restaurant feature extraction do exist, `www.citysearch.com` and `www.yelp.com` are prime examples. We believe, however, that between *Yahoo* and *Boorah*, we've managed to extract a large subset of the restaurants that exist within the Chicago, IL area. Though our dataset is sufficiently large to run analysis on, these additional sources may warrant research and extraction in the future.

2.3: Neighborhood Name Extraction

Both *Yahoo* and *Boorah* contain neighborhood classifications already associated with the restaurant feature. This made our extraction easy, since we didn't have to worry about parsing a description text field or attempt multiple or single matching. Some of the records from *Yahoo* and *Boorah* did not contain associated neighborhoods and were filtered out. After the filtering process, we obtained a total of 4,161 and 3,224 restaurant features respectively.

One challenge of neighborhood extraction for our restaurant features was that the neighborhood listed by *Yahoo* or *Boorah* did not always exactly match the list of neighborhoods we had obtained from the *Google Earth* KML (*e.g.*"Loop" instead of "The Loop").

Using some minor string parsing techniques and human error correction, we obtained a final list of 3,849 valid restaurant features for *Yahoo* and 2,990 for *Boorah*.

2.4: Geo-Coding Addresses

The process of geo-coding is to match an address to a latitude/longitude point on the globe. Unlike our apartment features, the restaurant features from *Yahoo* and *Boorah* did not come pre-geo-coded.

Google offers a geo-coding service as part of their *Maps* API, which we used match the street address associated with each restaurant feature to a latitude/longitude point [7]. The geo-coding process is simple: once we applied for a *Google Maps* API key, we sent each address to *Google*'s API and got a latitude/longitude point in return.

Using this API we were able to successfully geo-code 100% of our restaurant feature points.

2.5: Final Dataset

Since we created our restaurant feature dataset from two primary sources, we believed that there might exist a non-trivial percentage of duplicates within our database. In order to ensure unique records, we chose to group our listings on their latitude/longitude points as well as their associated neighborhood. In this way we could ensure that each restaurant feature was a distinct location or, if two restaurant features shared a latitude/longitude point, a distinct associated neighborhood. Using this grouping method, our final dataset contains 5,480 unique restaurant features.

6.2.3 Accuracy Analysis

At this stage in our analysis we were again only interested in a basic understanding of ground truth accuracy. We used the same point-in-polygon *winding algorithm* to match each restaurant feature to its actual neighborhood area as given by the *Google Earth* KMZ. Using this algorithm, we were able to match 4,945 restaurant features with an existing neighborhood boundary. We again postulate that the remaining 535 restaurant features fall outside of our given set of boundary areas.

Of the 4,945 restaurant features, we found that 49.32% matched our *winding algorithm* exactly. This percentage is much higher than our apartment feature's 31.8% match, which we postulate stems from our methods of neighborhood name extraction. Since we were able to extract the restaurant feature's associated neighborhood name directly rather than through text-field parsing, we believe the restaurants were more accurately classified.

Interestingly we did not see the same neighborhood-size relationship as with our apartment features. Within the restaurant feature dataset, the *Albany Park* neighborhood contained 223 distinct features with only a 34.52% accuracy, while the *Brighten Park* neighborhood, which is roughly the same size, contained 49 features with 100% accuracy (the *Magnificent Mile* did not contain any restaurant features and is unavailable for comparison). We believe that this once again has to do with the exact listing of neighborhood names within the restaurant feature dataset rather than the string parsing required with the apartment feature dataset.

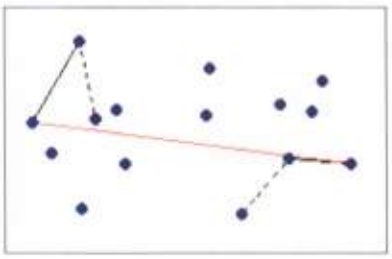


Figure 82: The opening stages of the Two Peasants algorithm.

Figure 83: A plot of the expected boundary (green) versus the extracted boundary (red) for Albany Park.

6.3 Visual Analysis

6.3.1 Visual Analysis

Since our goal with the neighborhood task is to represent neighborhood boundaries using the points in our dataset, we wanted a simple way to express the set of points as a polygon. At first we attempted to plot them in the order than we had collected each feature, but since this set is unordered, we ended up with a self-intersecting set of lines rather than a distinguishable boundary.

After some initial research, we settled on the "Two Peasants" algorithm for generating a simple polygon out of a set of points [26]. The algorithm is as follows:

1. Plot the points on an XY axis, where X-axis represents latitude and the Y-axis represents the longitude.
2. Locate the left most point (the point with the smallest X-value). Store this point as X_{min}
3. Locate the right most point (the point with the largest X-value). Store this point as X_{max}
4. Draw a line between X_{min} and X_{max}. Store its slope as m.
5. Initialize two empty arrays, *Top* and *Bottom*
6. For each remaining feature point, if the point lies "above" the line between X_{min} and X_{max}, place it in *Top*, otherwise place it in *Bottom*
   - In order to determine whether the point lies "above" or "below" the line:
     (a) Take the X-difference between X_{min} and the current feature point, P, store this as Δ.
     (b) Find the expected height, H, for the line between X_{min} and X_{max} at point P by $X_{min(y)} + (m * \Delta)$.
     (c) If $P_{(y)} > H$, place the feature point in *Top*, otherwise, place the point in *Bottom*.
7. Sort *Top* in increasing X values, sort *Bottom* in decreasing X values.

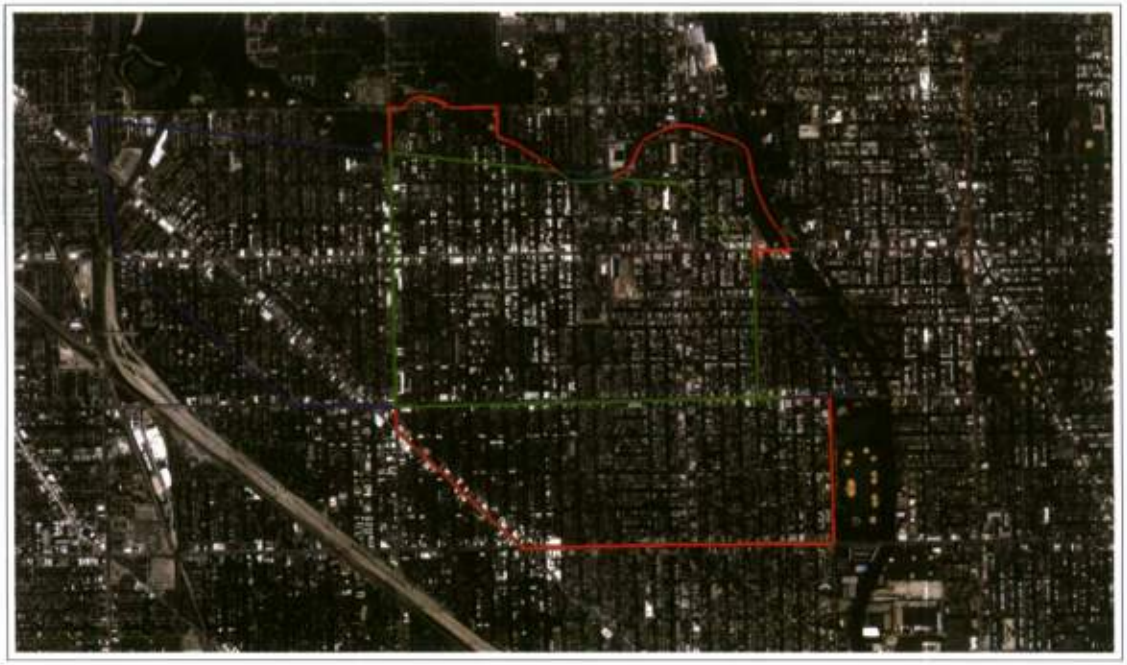


Figure 84: An overlay of polygons generated for Albany Park. The Convex Hull is shown in blue, the Ground Truth in red and their intersection in green.

8. Initialize an empty array, *Solution*
9. Place X_{min} in *Solution*, followed by each item in *Top*
10. Place X_{max} in *Solution*, followed by each item in *Bottom*
11. Since we're looking for a closed polygon, place X_{min} at the end of the array

Using this method, as illustrated in Figure 82, we draw a border in a clockwise manner around our set of feature points, obtaining a non-self-intersecting polygon which we can use as our neighborhood boundary.

We can plot both our expected boundary and the boundary obtained from the *Google Earth* KMZ for visual inspection. The downside of this method is that the boundary obtained from our "Two Peasants" algorithm includes every point in the generated boundary, even points that could be assumed to be on the interior of the polygon.

See figure 83 for a boundary plot on *Google Earth* in Chicago, IL.

6.3.2 Convex Hulls

The "Two-Peasants" algorithm uses all points in the dataset to create a closed, non-intersecting polygon. Although this proved better for visual inspection than simply examining the cluster of points, it includes points along the newly defined neighborhood boundaries that clearly lie on the interior of the polygon. The "Two Peasants" algorithm did not provide sufficiently smooth or regular boundaries for further analysis.

Our research identified another polygon generating algorithm, that of the "Convex Hull". This methods correctly identifies points which lie on the exterior of the polygon, those of the boundary, and points which lie on the interior of the polygon, which can be ignored in boundary generation. Using this algorithm, we are able to generate simple polygons which can be used in advanced analysis.

In order to determine the convex hull for each neighborhood feature set, we chose to use the Python language bindings for the popular OpenCV processing library [15]. All that is required to generate a convex hull is to provide an unordered list of points to the library's "ConvexHull2" method, which returns the set of clockwise-ordered boundary points that make up the polygon's convex hull. We ran this algorithm against each set of

neighborhood points for each of our two feature sets, apartments and restaurants, and stored the polygons in a database for later analysis.

6.4 Evaluation

6.4.1 Area Intersection

To refine our evaluation metric, we needed an algorithm that can compute the intersection of two polygons so that we can use the algorithm for evaluating the intersection of the convex hull generated above with the boundary suggested by the ground truth. We used a simple Polygon library written in Python [16]. This library extends the General Polygon Clipping Library, a widely known C polygon library written by Alan Murta, with additional methods written in Python.

Using the above area intersection algorithm, we generated the intersections of each of the convex hulls generated using our dataset with the corresponding ground truth. Since the neighborhood boundaries provided by our ground truth were already regular polygons, we were able to load them directly into the Polygon library by providing the same clockwise-ordered set of points that we extracted from our ground truth. We were then able to use the "Intersection" method (using the programmatic "&", *e.g.*"intersection polygon = polygon1 & polygon2") to define the polygon represented by the intersection of the given convex hull and its ground truth polygon.

Once we had the three polygons – the generated convex hull, the ground truth and their intersection – we were able to use the "Area" method from the Polygon library to determine the area of each.[4] As an illustration, Figure 84 shows these polygons generated for Albany Park.

6.4.2 Evaluation Metric

In order to assess the accuracy of the neighborhood boundaries generated by our algorithm, we defined a precision-recall based metric. We combined the precision and recall values into an overall "F1 Score" for each neighborhood.

Precision We defined precision of the generated neighborhood boundary as the ratio of the area of the intersection polygon to the area of the neighborhood boundary generated by our convex hull algorithm.

$$Precision = \frac{Area_{(Intersection)}}{Area_{(ConvexHull)}} \tag{4}$$

Since the area of the intersection can never be greater than the area of the convex hull, we know that if the areas are equal, and the ratio is 1, then the entire convex hull must reside within the ground truth. A ratio less than one indicates that $(1 - ratio) \cdot (100)\%$ of the convex hull lies outside of the ground truth, meaning that some of our extracted area is not accurate.

Recall We defined the *recall* of the generated neighborhood boundary as the ratio of the area of the intersection polygon to the area of the neighborhood boundary as suggested by the ground truth.

$$Recall = \frac{Area_{(Intersection)}}{Area_{(GroundTruth)}} \tag{5}$$

Since the area of the intersection can never be greater that the area of the ground truth, we know that if the areas are equal, and the ratio is 1, then the entire ground truth is covered by the intersection. A ratio less than one indicates that $(1 - ratio) \cdot (100)\%$ of the ground truth lies outside of our intersection, meaning it was not covered by our extraction.

F1 Score Finally, we combined the precision and recall metrics into the overall metric of F1 Score. The F1 Score is defined as the harmonic mean of the precision and recall metrics. Specifically, the following formula can be used to calculate the score:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{6}$$

[4]Note that the "Area" method given by the Python Polygon library assumes a Cartesian coordinate system and ignores the fact that our latitude and longitude points lie on a curved surface. However, since our analysis focuses primarily on the ratio between the convex hull, intersection and ground truth polygons, and since we use the same area method for each, this error is mostly negated. We also note that since our neighborhood areas are incredibly small compared to the earth's total surface, the curvature is negligible for our analysis.

Feature Type	Number of Areas	Precision	Recall	F1 Score
Apartments	64	32.74%	62.45%	.2847
Restaurants	83	56.28%	52.41%	.4310
Combined Dataset	101	36.31%	65.98%	.3330

Figure 85: Summary of the performance of the two dataset– Apartments and Restaurants, as well as the performance of the combined dataset. Each dataset could generate boundaries for a subset of neighborhoods, represented under "Number of Neighborhoods" column. The metrics of precision, recall and F1 score represent the performance of the respective dataset, when averaged over the neighborhoods for which the dataset could generate the boundaries.

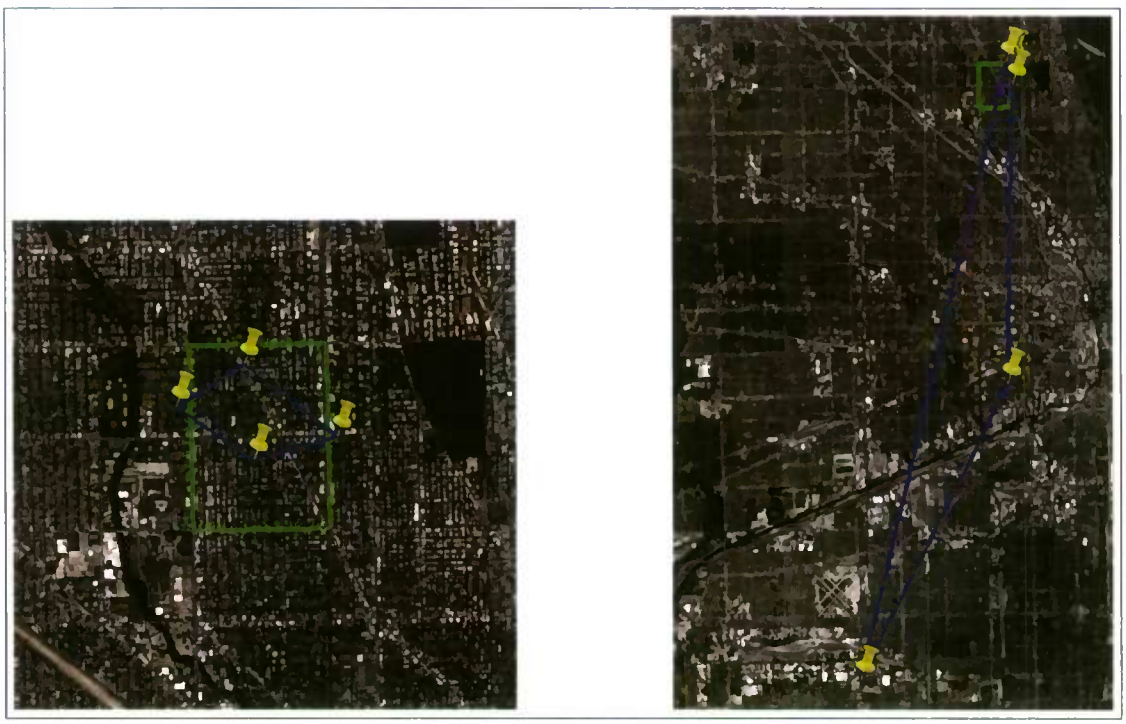


Figure 86: A *Google Earth* map of the North Center neighborhood comparing the convex hull representations of the apartment feature set (left) to the restaurant feature set (right).

6.4.3 Performance Results

The performance results for the apartment and restaurant feature sets are summarized in Figure 85. We find that our restaurant feature set provides a much better F1 score. This correlates with our initial findings that the point-in-polygon accuracy of restaurants was much higher than that of our apartment features. We also see, however, that the coverage of restaurant feature set is higher, that is, the apartment feature set could generate boundaries for 64 distinct neighborhoods, whereas our restaurant features cover 83 neighborhoods.

Additionally, we note that the low precision and recall scores may be due to limitations in our dataset. While the restaurant data tends to be more precise, restaurant features tend to fall only on major roads. This causes very linear formations of restaurant feature points once they are plotted on a map. The apartment data does not have this problem, since apartments may be spread throughout the city; however, the text extraction required by the neighborhood parsing causes additional natural language errors.

Figure 87: A *Google Earth* map of the Hyde Park neighborhood comparing the convex hull representations of the apartment feature set (left) to the restaurant feature set (right).

Precision Range	Apartments	Restaurants	Combined
0%	166	148	131
1 – 10%	17	9	26
11 – 20%	13	4	17
21 – 30%	5	5	7
31 – 40%	3	8	7
41 – 50%	4	3	7
51 – 60%	6	8	5
61 – 70%	3	11	7
71 – 80%	3	5	2
81 – 90%	1	4	6
91 – 100%	6	22	12

Figure 88: A binning of the precision scores from apartments, restaurants and the combination of the two.

6.5 Combination of feature sets

6.5.1 Combining Feature Sets

1: Motivation to combine the two dataset

It appears that there are multiple reasons that indicate that the two dataset – apartments and restaurants – may complement each other, *i.e.*, if combined together, the overall dataset may perform better than each of the dataset individually.

As our first reason, we believe that the two dataset when combined could generate the boundaries for a greater number of neighborhoods. We found that there were only 44 neighborhoods for which both the dataset generated boundaries; however, each dataset individually generated boundaries for a greater number of neighborhoods, *i.e.*, apartment dataset generated boundaries for 64 neighborhoods, and restaurant dataset generated boundaries for 83 neighborhoods.

As our second reason, we thought that having more data points for each neighborhood, after combining the

Recall Range	Apartments	Restaurants	Combined
0%	166	148	138
1 – 10%	5	4	5
11 – 20%	1	6	3
21 – 30%	3	5	3
31 – 40%	5	8	8
41 – 50%	9	15	9
51 – 60%	3	7	6
61 – 70%	5	10	9
71 – 80%	3	5	7
81 – 90%	5	9	11
91 – 100%	22	10	35

Figure 89: A binning of the recall scores from apartments, restaurants and the combination of the two.

F1-Score Range	Apartments	Restaurants	Combined
.00%	166	148	131
.01 – .10	11	6	15
.11 – .20	14	8	20
.21 – .30	11	7	10
.31 – .40	2	8	11
.41 – .50	10	12	9
.51 – .60	8	16	14
.61 – .70	5	9	9
.71 – .80	0	11	7
.81 – .90	0	2	1
.91 – 1.00	0	0	0

Figure 90: A binning of the F1 scores from apartments, restaurants and the combination of the two.

apartment and restaurant dataset, we have a higher chance of covering the entire neighborhood, *i.e.*, increasing the recall.

As our final reason, although the average performance of restaurant dataset is superior to that of apartment dataset, the apartment dataset performs better on a few neighborhoods.

We found that, although the average performance on F1 score is lower for apartment feature set when compared to restaurant dataset, there were 8 neighborhood instances where the F1 score of the apartment feature set was higher than that of the restaurant dataset. For example, the neighborhood "North Center," as shown in Figure 86, is one such area; the apartment feature set produces an F1 score of 0.4484 while the restaurant feature set is only 0.0361. We note from the *KMZ* map that the apartment feature set is fairly accurate whereas the restaurant feature set contains several points that are far from the ground truth.

On the other hand, we found 36 neighborhoods in which the restaurant feature set performed better than the apartment feature set. For example, the neighborhood of "Hyde Park" is one such area, as shown in Figure 87); the apartment feature set produces an F1 score of 0.1189 while the restaurant feature set is only 0.7507. We note from the *KMZ* map that the restaurant feature set is fairly accurate whereas the apartment feature set contains several points that are far from the ground truth.

In all of the cases, we observed that the major factor for our dataset giving a lower F1 score is due to significantly lower precision for that dataset, caused by points far from the ground truth boundary. Interestingly, we also observed that in many cases the recall of the dataset giving a lower F1 score feature is higher – much higher in the Hyde Park case–than the dataset giving a higher F1 score.

2: Performance of the combined dataset

We combined the two dataset – apartments and restaurants – to obtain a total of 7874 distinct feature points in the combined dataset. Due to the higher point-in-polygon accuracy of our apartment feature single match

approach, we decided to ignore the points extracted through multiple matching and only include the single match points with our neighborhood points. From this list, we kept the distinct set of features based on their latitude and longitude points as well as the expected neighborhood, which resulted in a total of 7874 features in the combined dataset.

As reported in Figure 85, the combined dataset generated the boundaries for 101 distinct neighborhoods – far greater than the number of neighborhoods for which either apartment or the restaurant dataset could generate the boundaries alone.

In the same figure, we also observe that the average F1 score of the combined dataset drops below the average F1 score of the restaurant dataset. This may be possible since performance of apartment dataset was superior to restaurant dataset only for 8 neighborhoods, while the restaurant dataset performance was superior for 36 neighborhoods.

To investigate further, we created a set of binning for precision, recall and F1 score across our three dataset (apartments, restaurants and their combination), as seen in Figure 88, 89 and 90, respectively.

In each bin, we noticed that the number of neighborhoods increase in the combined dataset as compared to the apartment or the restaurant dataset, indicating the improved coverage of the combined dataset with its greater number of points. Further, in Figure 88, which shows the binning results for precision, we notice that the number of neighborhoods increases most sharply in the $1 - 20\%$ bin, indicating the coverage of additional neighborhoods results in a reduction of precision. This is expected since an increase in the number of points increases the probability that many points will lie outside the ground truth boundary. Likewise, in Figure 89, we also see that the recall increases most sharply in the bin of $80 - 100\%$ – with additional points, the combined dataset has greater chance of covering the full neighborhood boundary as described by the ground truth.

We see that, in the F1 score bin, the combined dataset results in better performance specifically in the higher ranges when compared to the apartment dataset; however, in the higher range of F1 scores, combined dataset performs much worse than the restaurant dataset. Overall, we found that, in the combined dataset, only 17 neighborhoods increased their apartment F1 score, while 19 dropped. Likewise, only 18 neighborhoods increased their restaurant F1 score, while 35 dropped.

In summary, we found that the combined dataset produces higher recall for all neighborhoods, as well as that it can generate boundaries for a greater number of neighborhoods; however, this improved performance comes at the cost of lower precision, which results in a drop in the average F1 score for the combined dataset.

6.6 Outlier detection

6.6.1 Need for Detecting Outliers

We observed that, to improve the accuracy of the neighborhood boundaries, we must improve the precision of the generated boundaries. The combined dataset can give high recall – higher than 80% – for a large fraction, 46 neighborhoods out of the total of 101 neighborhoods, for which it could generate boundaries. In contrast, the combined dataset had low precision – lower than 20% – for 43 neighborhoods.

To improve the precision, we need to detect the far-off "outlier" points, which reside away from the core center of the cluster of "good" points. The reason for the lower precision is that a few datapoints lie far-off from the actual neighborhood boundary, and result in the convex hulls that cover a large area outside ground truth boundary. As illustrated in Figure 91, for the small Hyde Park neighborhood (which has been home to President Barack Obama, and is home to University of Chicago), our dataset contains a few points located far-off from the cluster of core points that correctly fall inside or close to the ground truth boundary.

6.6.2 Related Work

The problem of grouping a set of points and removing a few outlier points is related to various existing work in machine learning literature.

Clustering: There is a large body of research on clustering a set of points into homogeneous groups (*e.g.*, CLARANS [92], DBSCAN [80], BIRCH [93], STING [100], WaveCluster [69], DenClue [40], CLIQUE [83]). These algorithms typically require a pre-defined number of clusters or the threshold on the similarity metric as input, and are tuned towards partitioning the complete dataset into the desired number of clusters. For our scenario, we are looking for only one cluster as a result, so the solutions that partition the dataset into multiple clusters are not adequate.

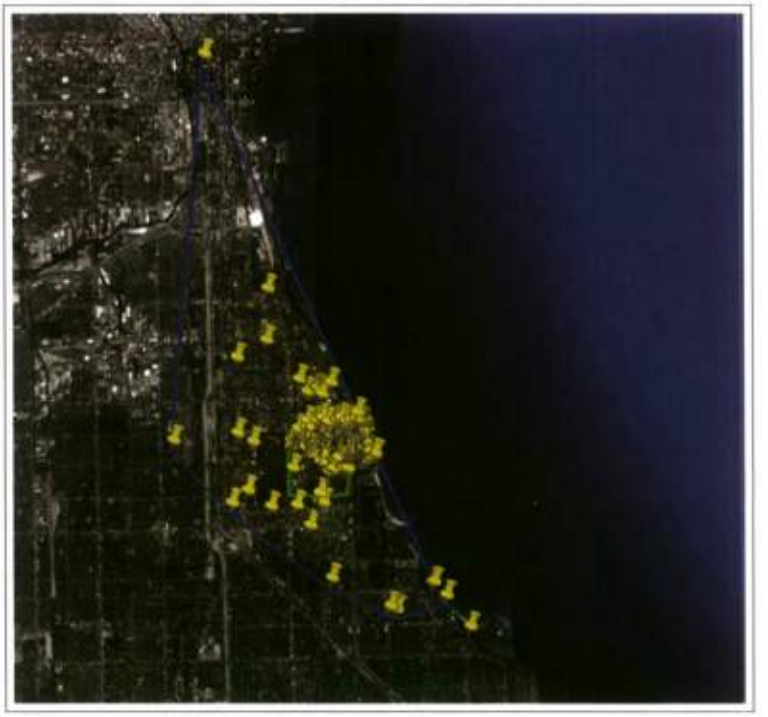


Figure 91: The Hyde Park neighborhood illustrates the need to detect far-off points to improve precision. The neighborhood boundary generated by our combined dataset (shown in blue) covers much larger area than the boundary of ground truth (shown in green).

Classification: A related set of techniques have been developed for classifying data points into multiple classes. These techniques generally require a labeled dataset for the training phase, and attempt to build a prediction model based on the properties of each point. For our problem, we do not have any features other than the coordinates of the points. Thus, each point by itself is not sufficient to determine if it should be kept in the analysis, or considered as an outlier point.

Outlier Detection: Several algorithms have been developed, especially in the statistics literature, for detecting points that appear to be "abnormal," *i.e.*, not following the general distribution such as Normal, Poisson, *etc.*. [97]. Many of these algorithms assume that the dataset comes from a certain underlying distribution, which does not hold true for the neighborhood dataset. Custom algorithms have also been developed for specific problems of fraud detection [91, 99]. These are not applicable to our scenarios of generating neighborhood boundaries. Some of the adaptations of the outlier detection work attempt to define the distribution of different properties of the data points, and predict the outliers as the ones following extreme values in one or more dimensions [50, 74]. These are again not applicable to our analysis as our data points have only two properties – latitude and longitude, which make the points indistinguishable without knowing the size of the neighborhood we are looking for.

Spatial Clustering: Perhaps the most relevant work to our problem is the *Clustr* algorithm [5], developed for the specific problem of generating neighborhood boundaries using tags of Flickr images. Unfortunately, the details of the algorithm are not available publicly. The source code was made open source, but it appears to be currently inaccessible.

Therefore, we decided to develop our own "Clustering" algorithms which will detect the central cluster of interesting data points while removing those outlier points that lie far-off.

6.6.3 Design Requirements for Clustering Algorithms

We faced three main challenges while designing our clustering algorithms. The first is that each algorithm must work across a very general, varied set of neighborhoods. That is, not every neighborhood will be identical in the

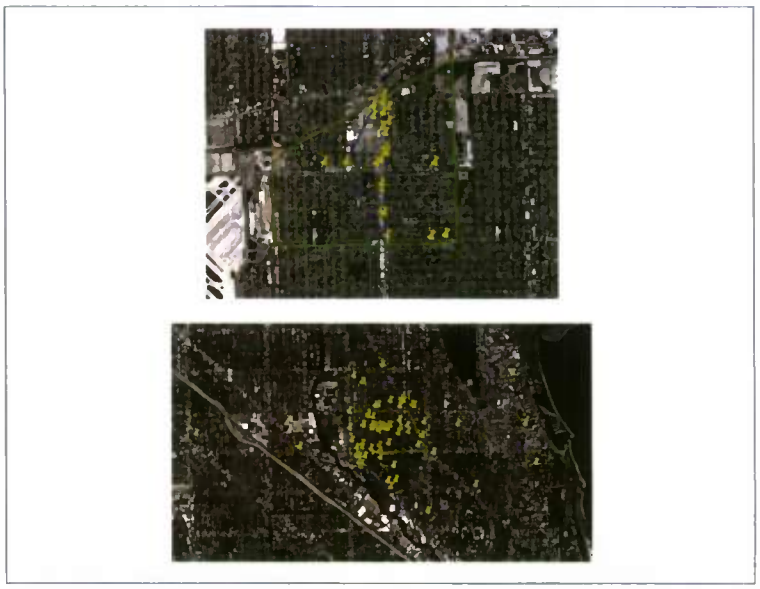


Figure 92: The West Elsdon (top) and Roscoe Village (bottom) neighborhoods. West Elsdon is an example of when Median Cluster works poorly, while Roscoe Village is an example of when Median Cluster works well. Yellow pins outside of the convex hull (blue) are points that have been removed. The ground truth is shown in green.

distribution of the "cluster + outlying points" shape – some contain two distinct clusters while others contain no discernible clusters whatsoever; furthermore, the points for some neighborhoods may already be in a homogeneous cluster, and so, should not require additional processing. Therefore, any algorithm that we design must either work positively on every configuration of points or at least work out more positively than it does negatively on an average.

The second challenge is that the usage of the ground truth should be limited to fine-tuning the parameters of the algorithm; however, the existence of such ground truth cannot be assumed for deployment scenarios. While we are using Chicago, IL as our target geography to develop the algorithms, the goal is to deploy these algorithms in geographies in which the ground truth is not available. Therefore, we cannot use the existing ground truth to make direct decisions about which points to exclude from our cluster core. For example, we cannot use statistics such as typical size, width, or height of the ground truth in our calculations. We can, however, use the ground truth and F1 scores to fine-tune the configuration parameters of our algorithms.

The third challenge is that, since all of our algorithms must focus on removing points from a neighborhood set, we must be careful to develop algorithms that can work successfully on relatively small sets of points. That is, an aggressive algorithm that removes large number of points is not so valuable, since, to begin with, some portions of our dataset contains small numbers of points for some of the neighborhoods.

Accounting for these challenges, we developed four different clustering and outlier detection algorithms, as described in following sections: "Median Cluster," "Nearby Cluster," "Circle Cluster" and "Negative Cluster."

6.6.4 Median Cluster

The idea behind the "Median Cluster" algorithm is that, given a distribution of points, all points in the "cluster" will be close together relative to the outlying points. Specifically, given a point, if the point exists within the "cluster", some percentage of other points should exist within the median distance of the whole point set.

The algorithm operates as follows:

1. Calculate the median of the distance between all pairs of the points.
    - That is, for each point, calculate the distance from it to each other point, sort these by length and find the median.

Threshold	Neighborhoods	Precision	Recall	F1 Score
10.00%	0	0%	0%	.00
20.00%	6	82.12%	0.11%	.0022
30.00%	68	68.04%	6.02%	.0981
40.00%	84	66.41%	18.88%	.2517
50.00%	84	62.39%	32.98%	.3723
60.00%	84	59.57%	43.47%	.4286
70.00%	84	56.71%	51.52%	.4599
80.00%	84	54.74%	57.90%	.4847
90.00%	84	49.71%	64.18%	.4786
100.00%	84	31.74%	74.08%	.3400

Figure 93: Summary of the performance of the Median Clustering algorithm at different values of threshold. For each threshold, the summary shows the number of neighborhoods that generated boundaries after applying the clustering algorithm, as well as the average performance with respect to precision, recall and F1 Score.

2. Initialize an empty set to hold all valid "cluster" points.
3. Establish a threshold percentage for which a point must have fewer distant neighbors relative to size of the neighborhood set.
4. For each point in the set:
   (a) Initialize a counter to zero.
   (b) Calculate the distance between the given point and each other point.
      - If the distance is higher than the median distance of the set of points, increase the counter by one.
   (c) After all points are checked, see if the ratio of counter to total points is less than the threshold percentage. If it is, add it to the valid cluster set.

Using this method, we find that the outlying points are generally removed. The drawback of this method is when our points are already clustered within the ground truth. We find that the "Median Cluster" algorithm will then remove the outlaying cluster points, which drastically reduces recall inside the neighborhood. As illustration, Figure 92 shows examples of neighborhoods in which Median Clustering algorithm performs well vs. not so well.

Since this algorithm requires an external variable – the threshold percentage for cluster inclusion – we set about discovering a way to determine the best value in a dataset agnostic way. Since our goal is ultimately to discover unknown neighborhood boundaries, and since we believe that, while the term for and definition of "neighborhood" may change from one locality to another, the general size should not. Therefore, we chose to test a number of thresholds against our current ground truth and select the best as the optimum "general" value. This way, though the selection of our threshold value is dependent on the ground truth, the distribution of points across all neighborhoods should serve to make our value selection general enough to work on any set of neighborhoods.

The performance of the Median Clustering algorithm across different threshold values is summarized in Figure 93. We observed that, for small values of threshold, *i.e.*, for situations in which we are expecting a large percentage of points to be far away, the number of neighborhoods for which the boundary could be generated is very small. As the threshold increases, we classify a greater number of points as valid cluster points, increasing our recall. At the same time, as additional points are included, the chance that a given point may be outside the ground truth increases, thus precision begins to drop. Eventually we reach an equilibrium between decreasing precision and increasing recall. At this point, our maximum F1 score is reached and the optimum value of the threshold is found. As the threshold increases further, precision begins to drop faster than the recall increases. We find the optimal threshold for the median clustering algorithm at roughly 80%, that is, for a given cluster point, 80% of the other points are allowed to be further than the median distance.

At the optimum threshold value, we are able to raise our average F1 score from 0.3330 for original dataset to 0.4847 using the Median Clustering algorithm. However, we are able to generate boundaries for only 84 neighborhoods using the Median Clustering algorithm instead of all 101 neighborhoods with original dataset. This is because many of our neighborhood sets are very small (4 or 5 points) to begin with and thus, once we begin removing additional points, these sets might become too small to properly create a convex hull.

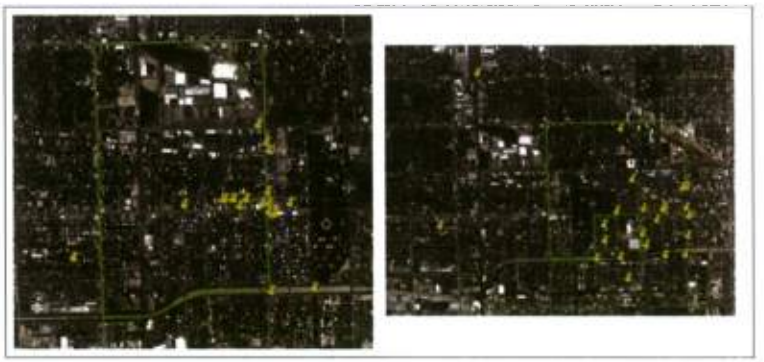


Figure 94: The West Garfield (left) and East Garfield (right) neighborhoods. West Garfield is an example of when Nearby Cluster works poorly, while East Garfield is an example of when Nearby Cluster works well. Yellow pins outside of the convex hull (blue) are points that have been removed. The ground truth is shown in green.

Threshold	Neighbors	Neighborhoods	Precision	Recall	F1 Score
120.00%	1.00	84	40.13%	69.63%	.4126
120.00%	2.00	84	44.80%	68.33%	.4483
120.00%	3.00	84	45.96%	66.56%	.4536
130.00%	1.00	84	39.95%	70.38%	.4163
130.00%	2.00	84	44.00%	69.05%	.4432
130.00%	3.00	84	45.56%	67.66%	.4553
140.00%	1.00	84	39.55%	70.74%	.4146
140.00%	2.00	84	43.54%	70.00%	.4457
140.00%	3.00	84	44.88%	68.65%	.4538

Figure 95: Summary of the performance of the Nearby Clustering algorithm at different values of threshold. For each threshold, the summary shows the number of neighborhoods that generated boundaries after applying the clustering algorithm, as well as the average performance with respect to precision, recall and F1 Score.

6.6.5 Nearby Cluster

The idea behind the "Nearby Cluster" algorithm is that, given a distribution of points, a point is more likely to be a valid cluster point if there are one or more points directly adjacent. That is, if a point lies by itself, it is reasonable to assume that it is an outlier and discard it from analysis.

The algorithm operates as follows:

1. Calculate the median of the distances between all pairs of points.
    - That is, for each point, calculate the distance from the point to each other point, sort these by distances and find the median.
2. Initialize an empty set to hold all valid "cluster" points.
3. Initialize two variables, a number of other points that must be "near" a given point and the ratio of the median distance that we designate as "near".
4. For each point in the set:
    (a) Initialize a counter to zero.
    (b) Calculate the distance between the given point and each other point.
        - If the distance is less than the designated ratio of the median distance, increase the counter.

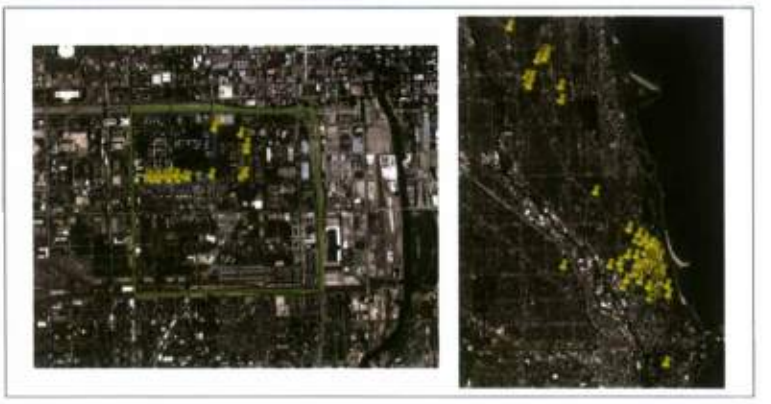


Figure 96: The University Village (left) and Old Town (right) neighborhoods. University Village is an example of when Circle Cluster works poorly, while Old Town is an example of when Circle Cluster works well. Yellow pins outside of the convex hull (blue) are points that have been removed. The ground truth is shown in green.

Threshold	Neighborhoods	Precision	Recall	F1 Score
10.00%	65	61.30%	30.88%	.2733
20.00%	84	51.87%	40.93%	.3200
30.00%	84	48.90%	52.40%	.3670
40.00%	84	45.12%	62.13%	.4060
50.00%	84	40.89%	67.60%	.4024
60.00%	84	38.93%	70.10%	.3942
70.00%	84	37.16%	71.81%	.3838
80.00%	84	34.74%	73.68%	.3646
90.00%	84	32.48%	74.75%	.3506
100.00%	84	31.74%	74.08%	.3400

Figure 97: Summary of the performance of the Circle Clustering algorithm at different values of threshold. For each threshold, the summary shows the number of neighborhoods that generated boundaries after applying the clustering algorithm, as well as the average performance with respect to precision, recall and F1 Score.

(c) If the counter is greater than or equal to the designated number, place it in the set of valid "cluster" points.

We rely on the "median" distance of points in the set to define "nearness," since we can't directly use our ground truth values. The drawback of this method is that several of our neighborhood sets contain fairly sparsely populated sets of data. As the number of points in a neighborhood set decreases, it becomes increasingly likely that the "Nearby Cluster" algorithm will begin to throw away points that should be valid. That is, if the core cluster is sparse, the algorithm may discard points too aggressively. As an illustration, Figure 94 shows examples of neighborhoods in which Nearby Clustering algorithm performs well vs. not so well.

We summarized the performance results in Figure 95 for subset of the combinations of the two configuration parameters – threshold on distance and the number of neighbors. We varied the number of neighbors from $1-5$, and the threshold on the distance from $10-200\%$. We found that the performance of this algorithm peaks when there are at least 3 neighbors for a given point. We also see that the optimum ratio of median distance is approximately 130%. This algorithm still performs better than our base combined data set, giving us a peak F1 score of 0.4553, but we note that it does not perform as well as the median algorithm.

6.6.6 Circle Cluster

The idea behind the "Circle Cluster" algorithm is that, given a large, dense cluster, the weighted center of the set of points should reside somewhere within or nearby the cluster. Using this assumption, we can include all of the

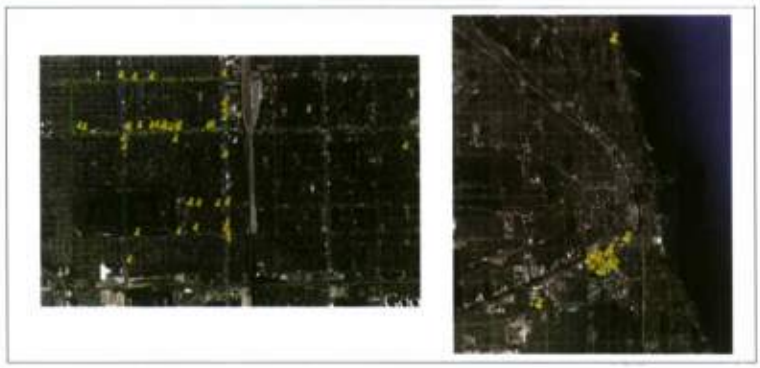


Figure 98: The Chicago Lawn (left) and McKinley Park (right) neighborhoods. Chicago Lawn is an example of when Negative Cluster works poorly, while McKinley Park is an example of when Negative Cluster works well. Yellow pins outside of the convex hull (blue) are points that have been removed. The ground truth is shown in green.

points within a certain radius of the weighted center as valid cluster points.

The algorithm operates as follows:

1. Calculate the median of the distances between all pairs of points.
    - That is, for each point, calculate the distance from the point to each other point, sort these by length and find the median.
2. Initialize an empty set to hold all valid "cluster" points.
3. Initialize a threshold percentage of the set's median distance from the weighted center where a point is considered "valid".
4. Calculate the weighted center of the set of points.
    - Take the sum of the X components and the sum of the Y components of all points in the set, divide by the size of the set.
5. For each point in the set:
    (a) Calculate the distance between the given point and the center point.
    (b) If the distance is less than the designated ratio of the median distance, place the point in the set of valid "cluster" points.

This method works very well for a neighborhood which contains tightly clustered points with outlier points reasonably far-off. However, if the outlier points are not far-off, the median distance of the set of points will be very small and the clustering algorithm will begin trimming valid points that exist very close to the center of the cluster. As an illustration, Figure 96 shows examples of neighborhoods in which Circle Clustering algorithm performs well vs. not so well.

We summarized the performance results of the Circle Clustering algorithm under different values of thresholds in Figure 97. We noticed that, although we maximize our F1 score at the threshold of 50%, the F1 score of the Circle Clustering algorithm tends to perform much worse than either the Median or Nearby clustering algorithms. We postulate that this is because a large portion of our neighborhood sets are already strongly clustered, thus we end up removing neighborhood points that should, in fact, remain valid.

Threshold	Percentage	Neighborhoods	Precision	Recall	F1 Score
10.00%	50.00%	84	42.07%	60.54%	.4043
10.00%	60.00%	84	40.95%	66.48%	.4121
10.00%	70.00%	84	39.31%	68.82%	.4009
20.00%	50.00%	81	48.40%	52.79%	.4099
20.00%	60.00%	82	47.06%	58.45%	.4327
20.00%	70.00%	82	44.17%	63.11%	.4223
30.00%	50.00%	78	52.61%	45.11%	.3974
30.00%	60.00%	82	50.25%	50.74%	.4103
30.00%	70.00%	82	47.87%	57.10%	.4297

Figure 99: Summary of the performance of the Negative Clustering algorithm at different values of threshold. For each threshold, the summary shows the number of neighborhoods that generated boundaries after applying the clustering algorithm, as well as the average performance with respect to precision, recall and F1 Score.

6.6.7 Negative Clustering

The idea behind the "Negative Clustering" algorithm is that, since our neighborhoods are adjacent to one another, an outlying point from a given neighborhood has a high probability of existing near a cluster from another neighborhood. Therefore, if a given point is near a large percentage of points which claim to be from a different neighborhood, it is probably an outlying point and can be marked as invalid.

The algorithm operates as follows:

1. Calculate the median of distances between all pairs of points.
    - That is, for each point, calculate the distance from it to each other point, sort these by length and find the median.
2. Initialize an empty set to hold all valid "cluster" points.
3. Initialize a threshold percentage of nearby non-set points where a given point will be considered invalid.
4. Initialize a percentage of the median set distance that will serve as "nearby."
5. For each point in the set:
    (a) Determine the number of points from the entire dataset which exist within the designated median distance ratio of the given point.
    (b) Determine the number of points from the entire dataset which exist within the designated median distance ratio of the given point and also claim to belong to another neighborhood.
    (c) If the ratio of points which claim to belong to a different neighborhood to the total points is less than the threshold percentage, place the point within the set of valid "cluster" points.

This method works well in almost all cases, as long as our dataset is dense enough to provide a reasonable degree of negative-example categorizing points. The benefit of this method is that it has a very low likelihood of removing valid points even in sparse neighborhoods, since it is generally unlikely that a valid point will be near a cluster of another neighborhood's points. The only scenario in which valid points are removed is when there is a single point on a neighborhood's border which is also close to a neighborhood with very dense borders. As an illustration, Figure 98 shows examples of neighborhoods in which Negative Clustering algorithm performs well vs. not so well.

We summarized the performance results of Negative Clustering algorithm under different values of thresholds in Figure 99. The algorithm consists of two configuration parameters – the threshold on median distance, and the ratio of points in different neighborhood to total points. We varied both the parameters from $0.1 - 1.0$, and presented the results in Figure 99 for a subset of the combinations. We found that the results from our Negative Clustering algorithm are fairly stable; it gives peak performance at 20% of median distance with a 60% invalid to total ratio, resulting in an F1 score of 0.4327.

Name	Advantages	Disadvantages	Best Use
Median Cluster	Works well when outlying data exists	If no outlying cases, algorithm may remove valid points	Strong clusters with some outlying points
Nearby Cluster	Works well with strong clusters	If data is spread evenly or very few points, algorithm may remove valid points	Strongly clustered data with some outliers
Circle Cluster	Works well when outlying data exists	If no outlying cases, algorithm may remove valid points	Strong clusters with outlying points
Negative Cluster	Works well in almost any dataset	Requires access to a significant amount of "negative" data	Filtering outlaying points based on negative examples

Figure 100: Summary of the different outlier detection algorithms.

6.6.8 Algorithm Chaining

We can see from Figure 100 that each outlier detection algorithm has advantages and disadvantages. We further postulated that it may be beneficial to use two or more clustering algorithms in tandem to achieve better results than we might with only one algorithm. We must, however, continue to balance the number of operations that are performed on a neighborhood set against the initial size of the set. That is, a smaller set will be able to support fewer operations before it no longer contains enough points to generate a valid convex hull.

With this in mind, we chose to limit our chain to two algorithms, specifically the Median Clustering and Negative Clustering operations. We chose Median Clustering because it performed the best of the three algorithms that work directly to identify the set of clustered points within a dataset. We believe that it is complemented well by Negative Clustering as the later has a smaller likelihood of removing valid data points.

Using the two algorithms in sequence – Median Clustering and Negative Clustering – with the optimum values of the configuration parameters as determined before, we were able to generate boundaries for 70 neighborhoods with an average F1 score of 0.5073.

6.6.9 Performance on Individual Dataset

We also wondered how well our clustering algorithms improve the performance of individual dataset. We generated the neighborhood boundaries for each of the apartment and restaurant datasets, after applying our Clustering algorithms. We found that our apartment feature set generated boundaries for 59 neighborhoods and improved the F1 score to 0.4158. Likewise, the restaurant dataset generated boundaries for 62 neighborhoods, while increasing the F1 score to 0.4819.

The impact of combining the two datasets is different across different neighborhoods. For the 59 neighborhoods for which the apartment dataset could generate boundaries, we saw that the combined dataset improved the F1 score for 31 cases, while it dropped the F1 score for 19 cases, and stayed at the same level for remainder of the cases. Likewise, out of the 62 neighborhoods for which the restaurant dataset could generate boundaries, the combined dataset improved the F1 score for 28 cases, while it dropped the F1 scores for another 16 cases, and stayed at the same level for remainder of the cases.

It is difficult to predict if the combined dataset will perform better or worse than the individual dataset for a given neighborhood; however, we found that the combined dataset can always generate boundaries for a greater number of neighborhoods and keeps the F1 score relatively high. So overall, a combined dataset will usually be more helpful.

6.7 Improving extraction methods

In an attempt to understand the natural language errors that may influence the accuracy of our neighborhood prediction within the apartment feature set, we assigned an analyst to examine 46 random feature points that

Total points inspected	46
Community Area	6
"Nearby"	7
Feature	16
Unambiguous	17

Figure 101: Breakdown of various types of ambiguity errors associated with the natural language parsing of Apartment feature descriptions.

were known, based on the point-in-polygon winding algorithm, to be outside of the ground truth.

To facilitate the inspection of the feature points, we placed these points, along with their description, address and expected neighborhood, on to a *Google Earth* KMZ map. We also included a map of the currently derived Chicago, IL, neighborhood boundaries. The breakdown of the different errors are summarized in Figure 101. We inspected each category of errors in more detail, as described in the following sections.

6.7.1 Resolving Extraction Ambiguities

1: Community Area Ambiguity

We postulated that, in some cases, an apartment's description could reference a community area rather than a neighborhood. Community areas are explicitly defined by a government agency in order to facilitate local organization of a city's resources. They are often larger than neighborhoods, many times including several distinct neighborhood areas within one community area.

Of the 46 neighborhood points examined, 6 share this type of ambiguity. Figure 102 shows an example of community area ambiguity. Of the 227 neighborhoods of Chicago, IL, 61 share the name with a community area, of which 51 neighborhoods are ones for which our combined algorithm generated the boundaries.

Unfortunately, apartment descriptions rarely distinguish "community area" or "neighborhood," so it would be very difficult to differentiate one type of description from the other. We cannot simply ignore all neighborhoods that share a community area name either, since the affected neighborhoods make up roughly 2/3 of all collected areas. It may be possible to collect polygons for each community area and exclude points that exist within the community area; however, community areas and neighborhoods of the same name often overlap. This might cause the removal of many valid points simply for the sake of excluding a couple of invalid ones.

In this case, it may only be that we know to be careful about the F1 scores generated by the shared-name neighborhoods.

2: Nearby Ambiguity

After analyzing neighborhood points, we found many that used "close by" terms such as "within walking distance," "nearby," and "minutes away from." These are inherently ambiguous terms which can range in distance from only a hundred yards to several miles, depending on the method of travel. Figure 103 shows one example of this type of ambiguity. The description for the example apartment feature reads, "A modern design 20 story high rise building featuring beautiful and natural surroundings of Lake Michigan and Rainbow Beach. Just minutes away from Hyde Park and downtown Chicago." We classified it as "Hyde Park" since no other neighborhood is mentioned. However, the apartment feature turns out to be nearly 2.5 miles away.

We believe it is possible to remove a subset of apartment features that contain this ambiguity. The problem is that there are too many "nearby" phrases, so it is difficult to enumerate them all, and capture all such invalid points. Also, if "nearby" really does mean within feet instead of miles, we may be removing a potentially valid apartment feature. It is also possible that we may be removing entries that use "nearby" language in reference to something other than a neighborhood. We prepared a list of nearby phrases, and matched them against our apartment dataset to determine their frequency, as summarized in Figure 104.

3: Feature Ambiguity We noticed that a neighborhood is sometimes named for a nearby geographic feature such as a park or local monument. This can lead to problems because, if the feature is sufficiently large, it can be bordered by several neighborhoods, any of which can contain apartment features that reference it. See figure 105 for one such example. This feature reports that it is close to "Harold Washington Park," which was selected as "Washington Park."

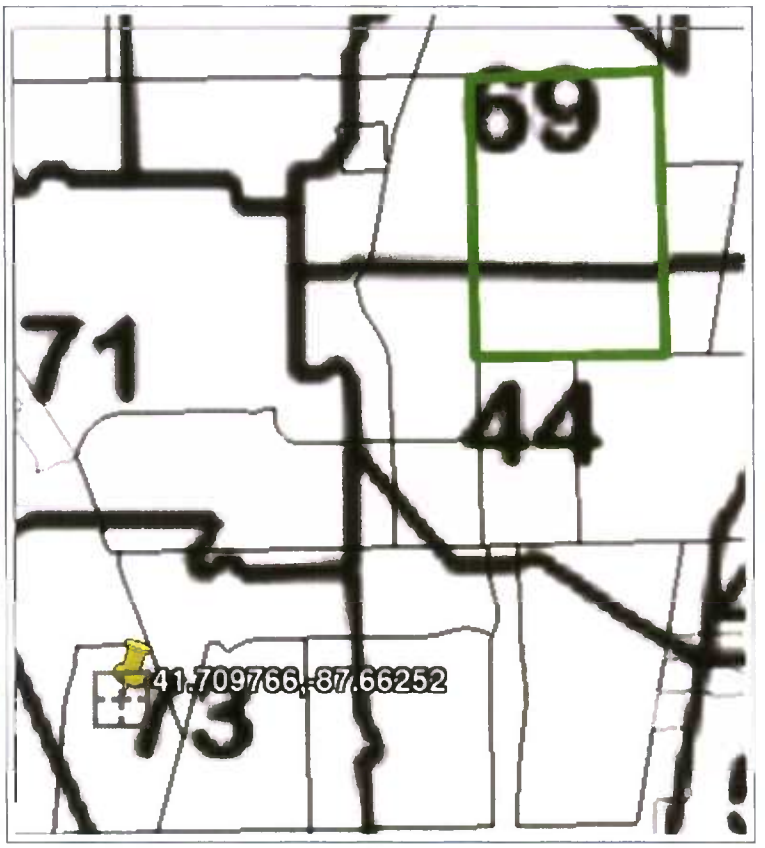


Figure 102: An example of ambiguous community area/neighborhood distinction. The apartment pin (yellow) reports that it's in the Chatham neighborhood (green). We can see the Chatham community area (#44 on the white overlay) is actually much closer, leading to possible ambiguity.

In addition to geographic features, we notice this same ambiguity with roads and highways. These are even more inaccurate because often the road does not exist anywhere near the reported neighborhood. For example, "Woodlawn Ave", "South Shore Dr.," and "Wrightwood Ave." are all roads that share their name with a neighborhood. Therefore, if an apartment feature holds the address in its description, *e.g.* "IMC Property Management Contact: 4521 S Woodlawn Ave Chicago, IL 60653-4407 Phone: 773-373-4300 Unit Type: Unspecified," we will select it as belonging to the "Woodlawn" neighborhood.

The last type of feature ambiguity is related to common phrases. We noticed that, for the "Lake View" neighborhood, we were picking up what seemed to be an abnormally high number of inaccurate apartment feature points. After analysis, we discovered that many apartment descriptions feature this phrase as a descriptor instead of a neighborhood, *e.g.* "...Spacious corner unit with city and lake views Plenty of living and storage space..." Of the 116 feature points that report the "Lake View" neighborhood, we found 51 that referenced the "lake views" plural descriptor. Of the remaining feature points, many still use "lake view" as descriptor, though since the descriptor and neighborhood name match exactly, it becomes extremely difficult to separate them.

It may be possible to cross reference a gazetteer of geographic features in order to remove the first type of feature ambiguity in the future. We were able to successfully remove 121 apartment features whose neighborhood was derived from their address rather than their description. In addition, we chose to remove the "Lake View" neighborhood entirely.

4: Unclassified / Unambiguous

We found that 17 of the 46 analyzed points were marked as "Unambiguous" by our analyst. These are apartment feature points whose descriptions seem to be unambiguously claiming to belong to a specific neighborhood and do not meet the criteria for inclusion with any of the other ambiguous-error categories.

Figure 103: An example of the "nearby" ambiguity. The apartment pin (yellow) reports that it is just "minutes away" from Hyde Park (red).

Phrase	Points Removed
"walking distance"	153
"nearby"	68
"minutes away"	26
"minutes from"	57
"close by"	24

Figure 104: A breakdown of various "nearby" phrase usage.

An example of one such description is, "Affordable apartments in South Shore Chicago, fully furnished with brand new appliances, hardwood floors, on-site laundry and more units with free Wi-Fi Internet. 24/7 security cameras to provide safety and a 24 hour maintenance hotline." We postulate that these entries are either incorrectly marked by their poster or, as may be the case in this specific entry, a set of several entries incorrectly marked with a very general apartment description.

5: Incomplete Address

We had previously developed a very simplistic method for detecting incomplete addresses, that is, if an address did not start with a numeric digit, we assumed it was incomplete. This removed entries such as, "N Western, West Side, Chicago, IL 60612" and "Orchard St, North Side, Chicago, IL 60614" which do not provide enough information to geo-code to a latitude/longitude point. In addition, we found that some apartment features had addresses such as "22nd Street, Chicago, IL" or "31st Street, Chicago, IL." These addresses were also invalid, but were not caught by the numeric digit check. We noted that these addresses are generally much shorter than a full address and, by filtering addresses with 22 characters or less, we were able to successfully remove 21 such entries.

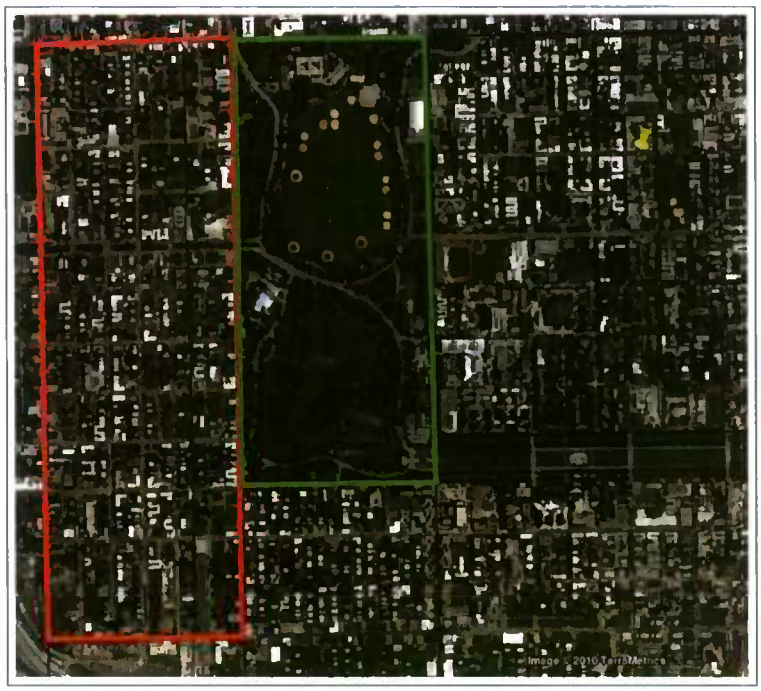


Figure 105: An example of the "feature" ambiguity. The apartment pin (yellow) reports "Harold Washington Park," (green), which was selected as "Washington Park" (red).

6.7.2 Performance Improvement from Removing Ambiguities

From our analysis of different reasons for extraction errors, and removing ambiguities as described above, we removed 471 of 2030 prior known apartment features. Upon this dataset, when we apply our neighborhood boundary generation algorithm, we obtain boundaries for 59 neighborhoods, with an average F1 score of 0.3126. The original dataset, without removing ambiguity, generated boundaries for 64 neighborhoods, with an average F1 score of 0.28. Thus, removing the ambiguity errors does result in an improvement in the performance. To put things into context, we should note that the original apartment dataset, followed by removing the outliers using our clustering algorithm, resulted in boundaries for 59 neighborhoods with an average F1 score of 0.41.

Overall, it seems that removing ambiguities linguistically leads to improved performance; however, our clustering methods yielded greater improvement in performance.

6.7.3 Targeted Extraction using Location Tokenization

As another tool to improve our extraction accuracy, we started initial exploration of a location tokenization technique. Our current method of neighborhood name extraction uses the full text from the description field of apartment features. This simplistic method results in choosing many data points with ambiguities that might be avoided with a more sophisticated approach.

To increase the precision of neighborhood name extraction, we decided that tokenizing the description string into lexical fragments would allow us to more precisely extract neighborhood names. After some initial research, we discovered `http://tagthe.net/` a website that, when fed a length of text, will break the text into various tags. `http://tagthe.net` even goes so far as to categorize the tags based on type, including a "location" section. We chose to use this "location" section of tags in order to extract the neighborhood name for a given apartment feature. See figure 106 for an example of an apartment description and set of returned tags.

Using an API provided by `tagthe.net` we were able to extract location-based tags for 2104 of our apartment features. Of these, we were able to discover single, valid neighborhood names for 412 apartment features. Note that the location-based tagging significantly reduces the number of apartment features that may contribute to generating neighborhood boundaries.

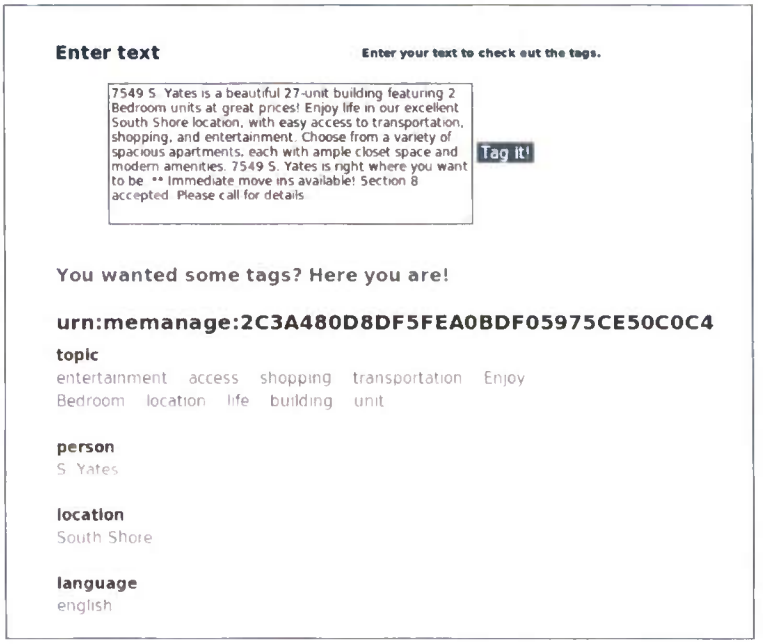


Figure 106: An example of the tags produced by `tagthe.net` for a given text string. We see that this apartment probably belongs to "South Shore."

We found that the apartment features obtained using location-tagging could generate boundaries for 17 neighborhoods, with an average F1 Score of 0.3362, much higher when compared to the base F1 score of 0.2847. The higher F1 score for this method indicates better extraction precision, since our extraction is now limited to the text classified as location tags, which can avoid several of the ambiguities in extraction.

We do note, however, that the location-tagged apartment features are far fewer, which results in a significantly smaller number of neighborhoods for which this method can generate boundaries. It may be possible to build our own string tokenizer/tagger with the ability to specify more specific known locations, in this case, for example, the Chicago, IL neighborhoods.

6.7.4 Gathering Additional Restaurant Data

We had previously found that the restaurant feature dataset provided higher accuracy than the apartment dataset, simply because the restaurant features were manually labeled into their respective neighborhoods – in contrast to our automatic text extraction for apartment features. In order to further improve the performance of our algorithms, we decided to gather additional restaurant features for our dataset.

1: Extraction of Restaurant Data

After an initial survey, we selected two additional sites for data extraction: `Zagat.com` and `OpenTable.com`. Both of these sources are well regarded in the restaurant industry. We believed that these would provide an excellent number of additional restaurant features for analysis.

Using our existing agent-based crawling ability, we were able to extract 1778 restaurant features from *Zagat* and 305 restaurant features from *OpenTable*. Note that some of these features specified neighborhood names that did not follow the same dictionary as our normalized ground truth dictionary. Therefore, we applied the "name matching" methods, as previously described, to match attempted the text-string "neighborhood" field of each feature to our database of known neighborhoods. As result, we were able to obtain 976 features from *Zagat*, and 175 features from *OpenTable*.

Of these, there were some duplicates within the same source of data as well. So, we de-duplicated the feature set from each source using latitude and longitude as well as the reported neighborhood feature. In this way, we remove the duplicates within each source of data. After this cleaning, we were left with 911 features from *Zagat* and 169 from *OpenTable*.

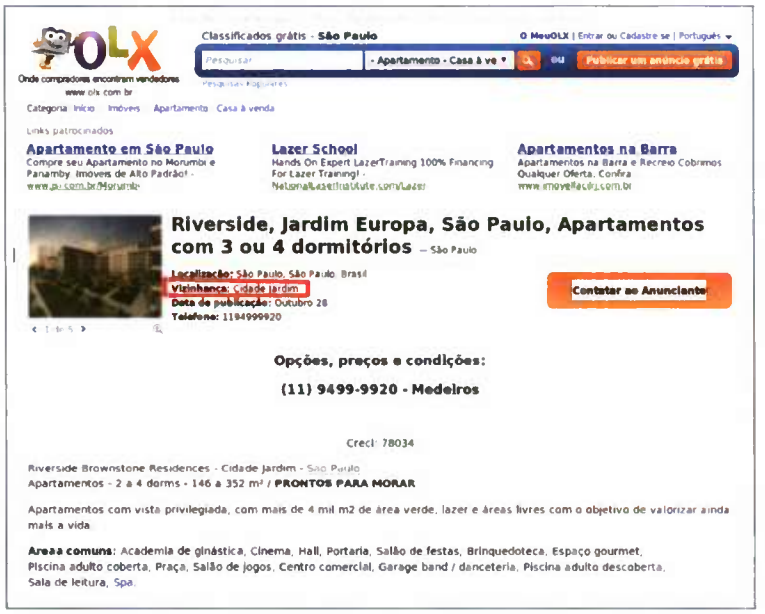


Figure 107: An apartment feature from *OLX*. The expected neighborhood is outlined in red.

2: Performance of New Restaurant Dataset

We used the new restaurant dataset to generate neighborhood boundaries and evaluated the performance against our ground truth boundaries. The data obtained from *Zagat* resulted in boundaries for 34 neighborhoods, with an average F1 score of 0.3159. There were far fewer features obtained from *OpenTable*, and as a result, it could generate boundaries for only 10 neighborhoods, with an average F1 score of 0.2517.

At first glance, these F1 scores seem fairly low compared to our previous restaurant feature sets. However, upon closer inspection we find that the drop in F1 score is due, in these cases, to low recall rather than low precision. The *Zagat* feature set holds a precision score of 42.05% and the *OpenTable* feature set holds a precision score of 62.87%. These are, in general, much higher than the precision scores of our previous apartment or restaurant dataset.

3: Performance of Combined Restaurant Dataset

We combined our newly extracted restaurant features with our existing dataset, in an attempt to improve performance. Our original restaurant dataset contained a total of 4827 features. With the addition of the new restaurant dataset, we added a total of 780 new restaurant features, resulting in a total of 5607 features in the combined restaurant dataset.

Together, the combined restaurant dataset could generate boundaries for 89 neighborhoods, with an average F1 score of 0.4157. This is an improvement in the number of neighborhoods over the original restaurant dataset, which could generate boundaries for 83 neighborhoods. The original restaurant dataset had a slightly higher F1 score of 0.4310.

4: Performance of Final Combined Dataset

Finally, we combined our entire apartment dataset after removing ambiguities, our old restaurant dataset, and the new restaurant dataset into a single final combined dataset. The new dataset, after applying our clustering algorithm, generated boundaries for 72 neighborhoods, with an average F1 score of 0.5018. This is an improvement in the number of neighborhoods over our prior best performing data set. Our old dataset, after clustering, generated boundaries for 70 neighborhoods. The F1 score of 0.5073 is comparable to the F1 score of the new method.

6.8 Foreign locality

6.8.1 Neighborhood Discovery for Foreign Country

1: Source Discovery and Data Extraction

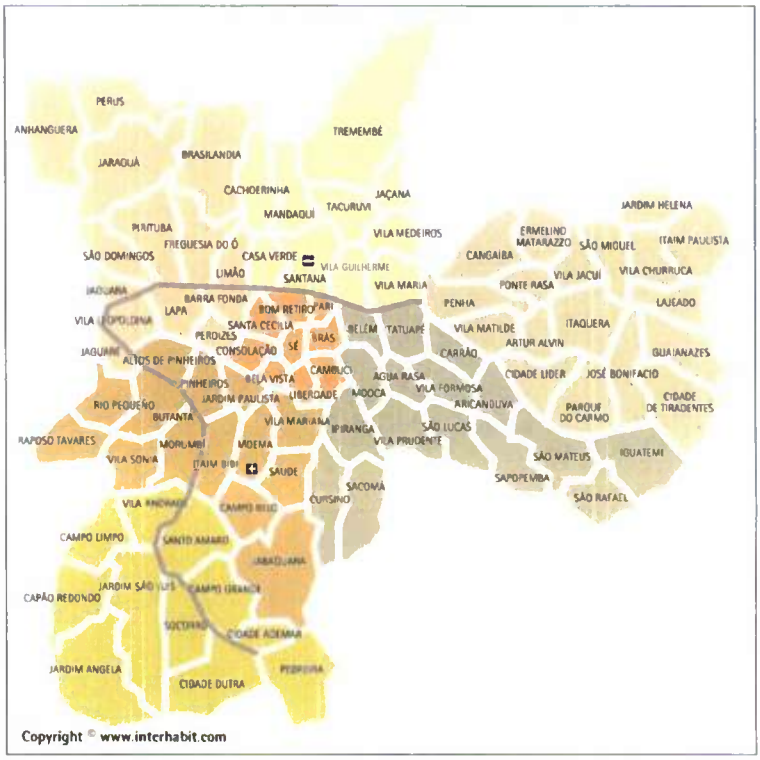


Figure 108: A map of neighborhoods in Sao Paulo provided by `interhabit.com`.

In order to further test our neighborhood extraction methods, we chose to attempt to derive neighborhood areas in an unknown, foreign locality. We were already examining apartment features from the Sao Paulo, Brazil, area for another project. We then decided to use Sao Paulo as our target foreign geography.

We discovered three primary sources of real estate listings in the Sao Paulo area – `http://cidadesaopaulo.olx.com.br/` (*OLX*), `http://www.aluguetemporada.com.br/` (*Alugue Temporada*) and `http://www.penseimoveis.com.br/` (*Pense Imoveis*).

These apartment sources, similar to the restaurant features discovered in Chicago, IL, contain an explicit "neighborhood" field. This means that we are not required to use any sort of string-parsing techniques and can instead define expected neighborhoods explicitly.

However, the information density of Sao Paulo, Brazil, is much lower than in Chicago, IL. Of our three primary sources, we only found apartment features specific to the Sao Paulo area from our *OLX* source. *Alugue Temporada* provided results for the *state* of Sao Paulo but did not contain any results for Sao Paulo city; *Pense Imoveis* did not seem to contain many listings with neighborhood information. As an illustration, Figure 107 gives an example of an apartment feature from *OLX*.

In all, from *OLX* we were able to extract 1246 records with the neighborhood field available.

2: GeoCoding of Addresses

The extracted apartment features did not provide an explicit latitude/longitude coordinate for mapping. We used *Google Maps* service for geo-coding, and, using this method, we were able to define geo-coordinates for all 541 apartment features.

To remove duplicate points, we grouped our apartment features based on their latitude, longitude and reported neighborhood. Of the 541 original apartment features, we found 370 distinct features.

3: Boundary Generation

We chose to test our general cluster analysis on the apartment features extracted from Sao Paulo. We used the same analysis as in our previous experiments. That is, we chained the "Median Cluster" and "Negative Cluster" algorithms together with their optimum input parameters as discovered from the Chicago, IL locality.

Figure 109: A map overlay of the extracted "Vila Maria" neighborhood after cluster analysis. Our predicted convex hull boundary is seen in red.

After applying clustering, the remaining feature points could generate boundaries for six neighborhoods in Sao Paulo city: "Brooklin," "Ipiranga," "Morumbi," "Saúde," "Tatuapé" and "Vila Maria."

6.8.2 Ground Truth Discovery

Using our "downward" *Wikipedia* extraction from the root category, "Districts in Sao Paulo" (`http://en.wikipedia.org/wiki/Category:Districts_of_S%C3%A3o_Paulo`, we located 59 distinct neighborhood names.

We were unable to find a ground truth that would provide the boundaries for each neighborhood, and so were unable to evaluate the performance using our evaluation metrics of precision, recall and F1 scores.

We did, however, find a current map of expected neighborhood boundaries, as shown in Figure 108. We can use the map for visual inspection of the neighborhood boundaries generated by our algorithm.

1: Normalization of Neighborhood Names

We found that the neighborhood names in our dataset were not standardized. From *OLX* we were able to extract 1246 records. Since our apartment features contained an explicit "neighborhood" field, we were not required to use a tokenizer or string parser to define each neighborhood. However, the neighborhood field of *OLX* does not seem to be standardized. When grouped by neighborhood name, we find that the 1246 provide 521 distinct neighborhood names; as a reference, our ground truth discovery using *Wikipedia* indicates there are a total of 59 neighborhoods in Sao Paulo.

We believe that this is because the "neighborhood" field of *OLX* is entered by hand. Small differences in neighborhood name conventions mean that we must provide some normalization in order to extract a reasonable number of neighborhoods for analysis.

We developed a very simple string-parsing method in order to match as many names to our ground truth as possible. For each known neighborhood name, we examined each extracted neighborhood to see if the known name existed anywhere within the extracted name. In this way, we were able to match 541 apartment features to known neighborhoods.

6.8.3 Performance Evaluation

Since we were unable to obtain ground truth boundaries for neighborhoods in Sao Paulo, we instead used visual inspection to evaluate the accuracy. Figure 109 shows the overlay for the "Vila Maria" neighborhood. From the visual inspection, the accuracy appears to be good; based on visual inspection, we estimate the precision to be 70% and recall to be 60% for this neighborhood.

7 Wikipedia Traversal

In this task, we began an investigation that generated gazetteer entries using content available on Wikipedia. We observed that many pages on Wikipedia include geo-coordinates, specifically the articles about geospatial features, *e.g.*, mountains, lakes, cities, hospitals, buildings and other structures, *etc.* Having a geospatial database constructed from Wikipedia, and composed of the rich human-edited content together combined with associated coordinates, would be an immensely useful resource for driving many mining tasks. The coordinates on the Wikipedia page are available in a consistent format, as shown in Figure 110. While the actual position in which the coordinates are placed may differ across pages, the formatting properties are always consistent.

Additionally, we explored an structured alternative to *Wikipedia*, *DBPedia*, as well as traversal techniques for acting on *Wikipedia* articles, feature gazetteer generation and article autocorrection.

7.1 Coordinate discovery

7.1.1 Motivation

Transitioning from our study of discovery of the hospital features, we started this task with a similar focus of constructing a database of hospitals and their coordinates using Wikipedia. We built agents with which to extract the coordinates from all the articles on Wikipedia about hospitals in the United States. Our agents started from the page on Wikipedia that provides a "List of hospitals in the United States" (`http://en.Wikipedia.org/wiki/List_of_hospitals_in_the_United_States`). From this root page, our agents visited pages corresponding to each state, and collected the URLs of individual hospital pages. Finally, upon visiting these individual hospital pages, our agents extracted key attributes, including the name of the hospital, and its coordinates (whenever available).

In our opening section, we describe the overall motivation for our exploration of generating geospatial gazetteers from novel forms of geo-tagged media content. The knowledge-rich media content, when geo-tagged, can become a powerful resource for mining. The text in Wikipedia articles is an example of such rich content. Recently, there has been an increasing buzz around the real time messaging service *Twitter* for its planned release of a geo-tagging function [38]. This service is already receiving much attention for its capability of enabling real-time communication on the Web. The service is now considered akin to *citizen journalism* [36, 37]: for example, the news about the plane crash in the Hudson River [24] was reported on *Twitter* before the national news agencies picked up the story. Similarly, live updates of a house on fire [12], or the attacks in Mumbai [28] first appeared via *Twitter*. Now, imagine: what if all these messages were geo-tagged? They could become a great resource for the mining of intelligence information.

We observed that many pages on Wikipedia include geo-coordinates, specifically, articles about geospatial features, *e.g.*, mountains, lakes, cities, hospitals, buildings and structures, *etc.* Constructing a geospatial database based on rich human-edited content of Wikipedia articles, together with these geo-coordinates, could provide an immensely useful resource for driving many mining tasks. One function in Google Maps illustrates one of these numerous possibilities. As shown in the screenshot in Figure 111, the service displays the Wikipedia articles as a layover on the map. Users can navigate the map to browse to their favorite location and read the articles related to a geography pertaining to their interest.

7.1.2 Extraction of Coordinates from Wikipedia

1: Basic Extraction

We built agents to extract coordinates from Wikipedia articles about hospitals, as illustrated in Figure 110. The agents collected a total of 983 hospital features. The agents began from the page on Wikipedia providing a "List of hospitals in the United States" (`http://en.wikipedia.org/wiki/List_of_hospitals_in_the_United_States`). Note that while we used the hospitals in the United States as the corpus for our study, our goal was to

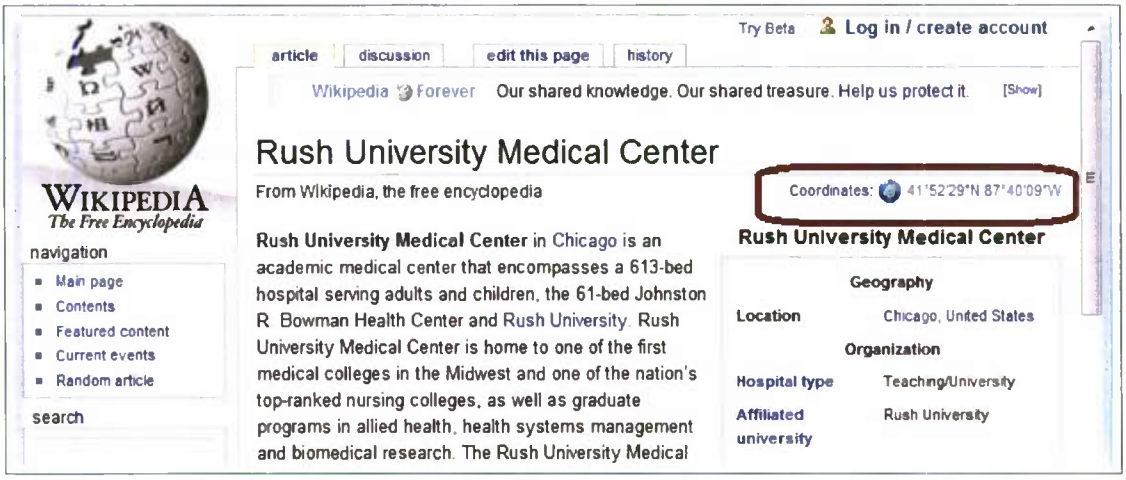


Figure 110: Illustration of Wikipedia articles that include geocoordinates.

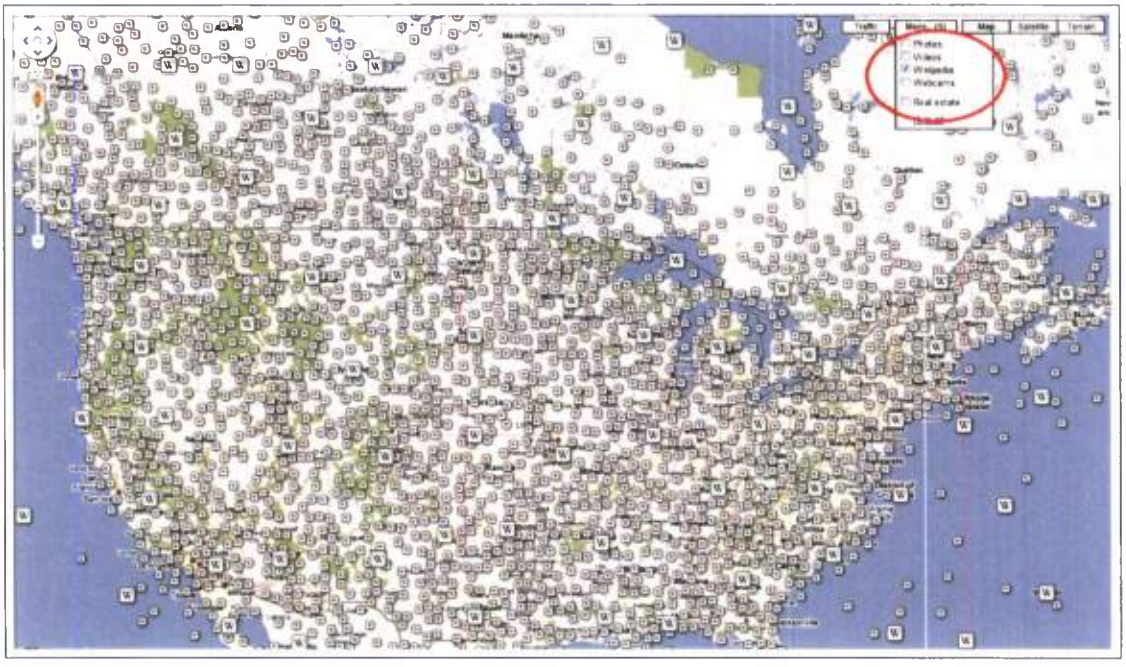


Figure 111: Layover of Wikipedia articles on Google Maps.

design techniques that could automatically discover coordinates for all geographical features in Wikipedia, *e.g.*, buildings, airports, amusement parks, bridges, canals, dams, cinemas, *etc.* (`http://en.wikipedia.org/wiki/Category:Buildings_and_structures_in_the_United_States_by_state`).

2: Discovery of Coordinates: UsingName Method

In our first method for automatic discovery of coordinates, we used solely the hospital name to compute the geo-coordinates. Often, third party geocoding services can return correct coordinates based only on the name of a hospital. We sent the #hospital-name as our input query to the *Google Maps* geocoding service. The resulting coordinates were used as the output of the UsingName method.

To understand the correctness of the UsingName method, we compared its output coordinates to the coordinates obtained from the Wikipedia articles. In Figure 112, we show the distribution of the distance between

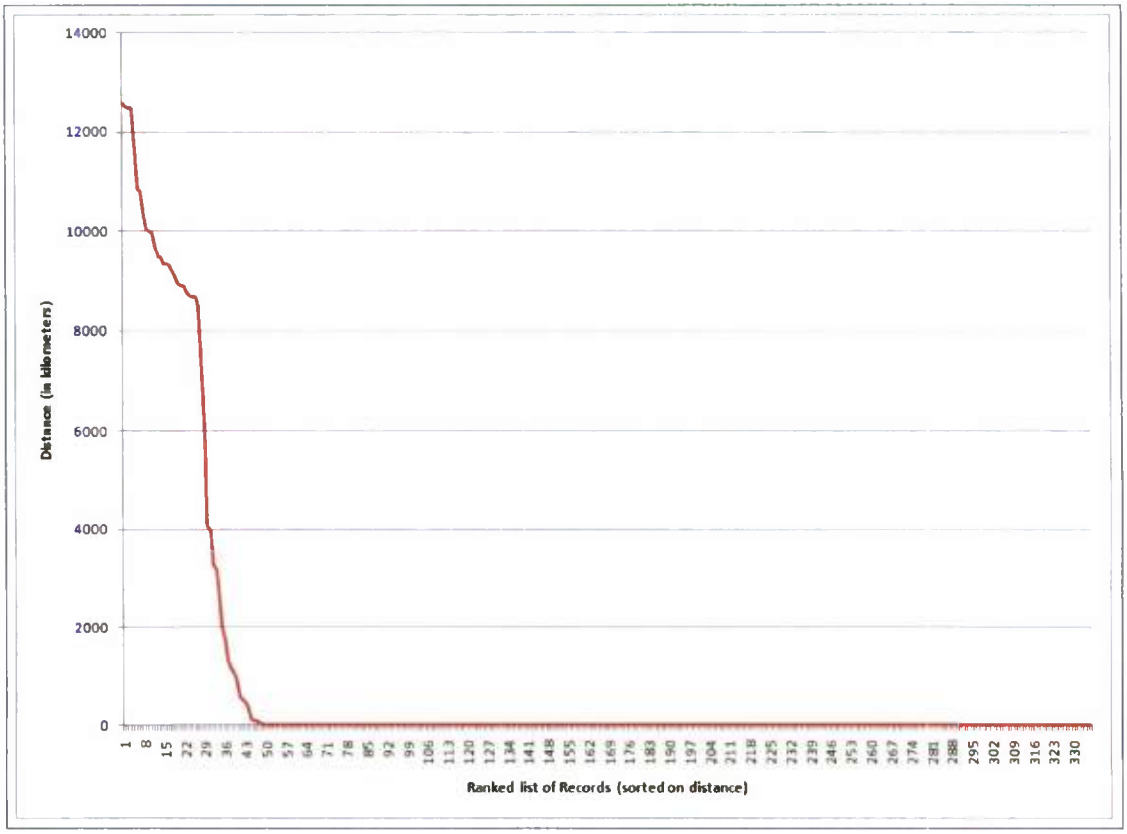


Figure 112: Distribution of the haversine distance between the coordinates from Wikipedia and the UsingName method.

the coordinates obtained from the UsingName method and from Wikipedia, for the 382 hospitals for which the Wikipedia articles had coordinates. We observed that for some cases the distance is too far–even greater than the distance between *Chicago* to *New Delhi, India*, *i.e.*, 12,000 kilometers.

To improve the precision of the UsingName method, we added a module to analyze the coordinates returned by the geocoding service and to filter out the "error" cases, for which the results of this method cannot be correct. We attempted to filter out the following error cases:

1. Error: Cannot Geocode - For some cases, when the third party geocoding service (*i.e.*, *Google Maps*) failed to geocode the input query string of #hospital-name, the service returned the default coordinates of 0, 0. For these cases, we modified our UsingName method to return a 'null' result (*i.e.*, cannot geocode).

2. Error: Outside US - Since our target geography in this study was the United States, for the cases in which the geocodes obtained from the geocoding services were outside the United States, we modified our UsingName method to again return 'null' results.

3. Error: City Level - Generally, the geocoding services attempt to return the best possible match for the input query, and as a result, sometimes their results are at the granularity level of the city. These were, again, not precise enough for our purpose, as we were looking for a precise coordinate of the feature (*e.g.*, hospital). We modified our UsingName method to detect such cases by adding a function that analyzes the "place" structure (*i.e.*, the normalized address string) returned by the geocoding services and determines if the most specific field in the normalized address is a city.

After filtering out the erroneous cases, as detected by the above modifications, we found that the UsingName method returned coordinates for 765 cases. We considered these cases "seemingly okay," *i.e.*, we believed these

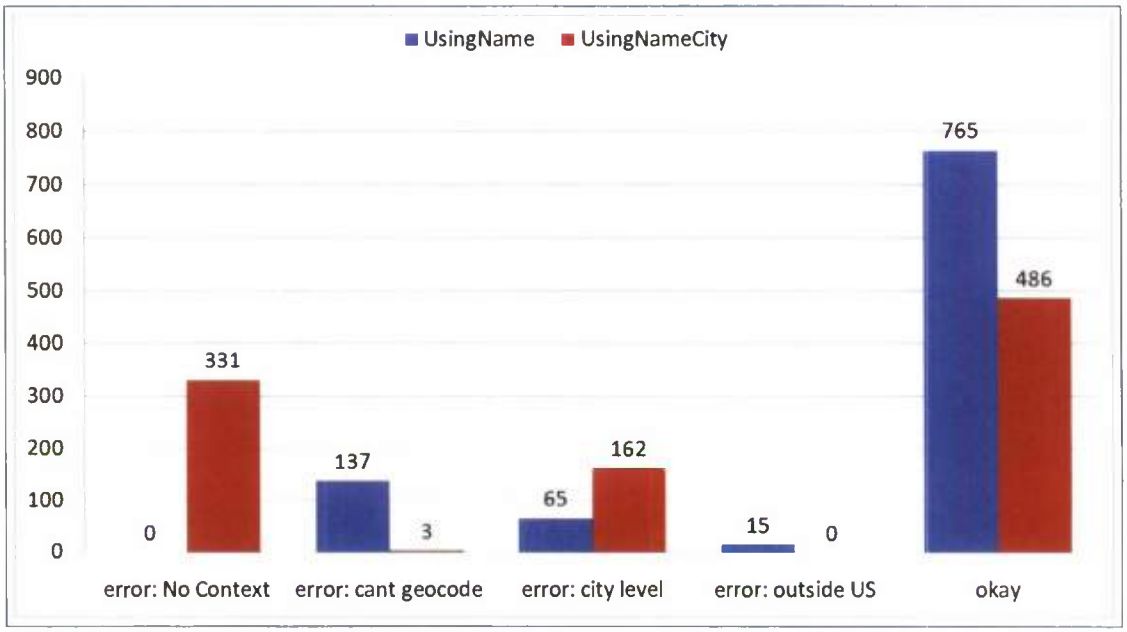


Figure 113: Breakdown of the results of the UsingName and UsingCity methods into different "error" cases and "seemingly okay" cases.

coordinates should be correct. As summarized in Figure 113, the results of the UsingName method fall under "Error: Cannot Geocode" category for 137 cases, "Error: Outside US" for 15 cases, and under "Error: City Level" for 65 cases. Thus, the UsingName method successfully returned results for 765 out of the 983 total hospitals on Wikipedia, *i.e.*, the recall of the UsingName method was 77.8%.

The distribution of the haversine distances between the coordinates from Wikipedia and the coordinates from the UsingName method appeared less skewed after filtering out the "error" cases. Of the 765 cases in which the UsingName method returns coordinates, Wikipedia coordinates were available in 288 cases. The distribution of the haversine distances for these 288 cases is shown in Figure 114. In comparison with the distribution before filtering (*i.e.*, Figure 112), far fewer cases showed high distances after filtering; most cases were closer to 0.

We also evaluated the precision of the UsingName method. We sampled hospital features from 288 cases in which both the UsingName method and the Wikipedia coordinates were available. We partitioned these hospital features into different groups based on the distance between the two coordinates, as shown in Figure 115. For each partition we sampled up to 10 hospital features. For the sampled hospital feature we manually inspected to determine whether the hospital is indeed located at the returned coordinate. We observed that the precision for the cases in which the distance is 0 kilometers or less than 0.5 kilometers was 100%. The precision dropped to 66% for distances between $0.5 - 1$ kilometers, and then to 57% for distances in the $1 - 10$ kilometer range. Precision dropped to 0 for distances of more than 10 kilometers. Overall we estimated that the coordinates returned by the UsingName are correct in 93.4% of all cases.

Thus, the precision and recall of the UsingName method are 93.4% and 77.8%, respectively.

3: Discovery of Coordinates: UsingCity Method

Next, we designed a second method that could outperform the UsingName method. We added some context to the name of a given hospital before geocoding. In particular, in our second method we used the additional context of providing the name of a city and state in the hospital feature. Thus, we called our second method the UsingCity method. This additional context helped the geocoding service to better compute the correct coordinates for the input query.

As the first challenge, we had to extract the city name from the content of a Wikipedia article. Consistent with the motivation of this study–that Wikipedia articles provide rich knowledge about the respective hospital feature–we observed that the articles generally included the name of the city in which the hospital was located, often in the first paragraph of the article. Furthermore, the mention of a city name is *hyper-linked* to the Wikipedia

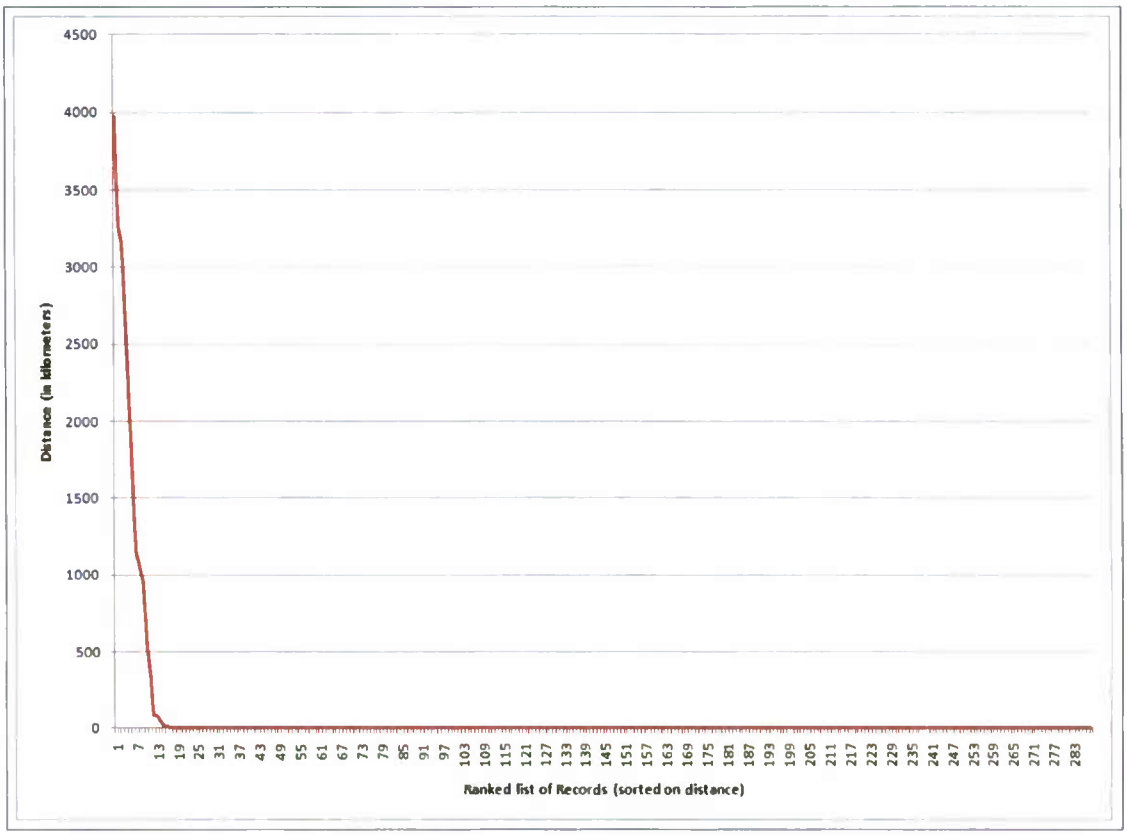


Figure 114: Distribution of the haversine distance between the coordinates from Wikipedia and the UsingName method, after filtering out the error cases.

Distance	Records	Sampled	Correct	Estimated Correct	Precision
0	69	10	10	69	100%
$0 - 0.5$	194	10	10	194	100%
$0.5 - 1$	3	2	2	2	66.67%
$1 - 10$	7	7	4	4	57.14%
$10 - 100$	4	4	0	0	0%
$100 - 500$	1	1	0	0	0%
$500 - 1000$	2	2	0	0	0%
> 1000	8	8	0	0	0%
Total	288			269	93.4%

Figure 115: Evaluation of the precision of UsingName method.

article about that city. The anchor text of these hyper-links (*i.e.*, the text used in an *hyper-link* to another web page) were consistently of the format "#city, #state."

Based on our observations, to extract a city name, we first extracted all the *anchor text* from the first paragraph of a Wikipedia article. We, then, matched each of the candidate anchor texts with a "#city, #state" pattern to determine which of them end with the name of any of the state of the United States. We used the first matching anchor text, if any, as the city context for that hospital.

Next, after extracting the city context, we used the string "#hospital-name #city-context" as the input query for the geocoding service of *Google Maps*. We used the returned coordinate from the geocoding service as the result of the UsingCity method.

Using our technique for extracting the city name, we were able to find the context in 652 out of 983 hospital

Distance	Records	Sampled	Correct	Est. Precision
0	15	9	9	100%
$0 - 0.5$	172	10	10	100%
$0.5 - 1$	7	7	7	100%
> 1	8	8	4	50%
Total	202			98%

Figure 116: Evaluation of the precision of the coordinates obtained from the UsingCity method.

articles listed on Wikipedia. For the pages in which our method returned the city context, we observed that the results were always correct. For the remainder pages in which our method could not find the city context, *i.e.*, 331 out of 983 pages, we observed that the reason for this result was due to the missing state name in the anchor text. Our city context extraction searched for the state name in the anchor text; however, the articles about hospitals in large cities, *e.g.*, Chicago, seemed to only mention the city name, without the suffix of the state name. For example, the article about "John H. Stroger Hospital" used the anchor text of "Chicago" (`http://en.Wikipedia.org/wiki/John_H._Stroger,_Jr._Hospital_of_Cook_County`). These cases can be improved by making our city context extraction flexible, by relaxing the requirement that state name must appear in the anchor text. We can add another dictionary of all cities or "popular" cities, and look for matches against the city name dictionary.

We analyzed the accuracy of the results of the UsingCity method by employing a framework similar to the one we used for the UsingName method. As shown in Figure 113, there were 331 cases for which the city context could not be found. We observed that adding the city context helped improve geocoding accuracy: the number of cases of "Error: Can't Geocode" fell to only 3 with the UsingCity method, in comparison to 137 for the UsingName method. Also, there were zero cases for which the geocoded addresses fell outside the United States with the UsingCity method; in contrast, with the UsingName method, there were 15 such cases. We found that the number of cases for which the geocoded address was at the granularity of the city level, *i.e.*, the category of "Error: City Level," was higher for the UsingCity method–162 cases, compared to only 65 for the UsingName method. Since the UsingCity method used the city context, the geocoding service was likely to return a city level address in those cases for which it could not find the geo-coordinates for a specific hospital name.

Thus, the UsingCity method could successfully geocode 486 out of 983 hospital records, *i.e.*, a recall of 49.4%. The recall of the UsingCity method is significantly lower than the success of the UsingName method, which was 77.8%.

Next we evaluated the precision of the UsingCity method. For our evaluation we used a similar approach employed in the evaluation of the UsingName method, namely, by sampling records based on the distances between the coordinates of the UsingCity method and those coordinates obtained directly from Wikipedia. Recall that we already did such sampling when evaluating the precision of Wikipedia coordinates (Figure 121). We used the same sample set of records for evaluating the precision of the UsingCity method as well, as summarized in Figure 116. There were 15 records for which the distance between the two coordinates was zero, *i.e.*, the coordinates were exactly the same. Of these we sampled 9 records, and in all cases we found that the coordinates of the UsingCity method were correct. For the next partition, in which the distance between two coordinates was between $0 - 0.5$ kilometers, there were 172 records; we sampled 10 records and found the precision of the UsingCity method to be 100%. Moving to the next partition of distance, between $0.5 - 1$ kilometers, we again found the precision of the UsingCity method to be 100%. For the last partition of distance > 1 kilometers, the precision of the UsingCity method dropped to 50%. Overall, the precision of the UsingCity method was estimated to be 98%.

7.1.3 Evaluation of Coordinates Extracted from Wikipedia

We first studied the accuracy of coordinates available in Wikipedia articles by analyzing the results produced by our agents.

We observed that on one hand the hospital articles found on Wikipedia were often missing the geo-coordinates. Only 335 out of 983, *i.e.*, 34% of the hospital articles on Wikipedia, included geo-coordinates.

On the other hand, when we inspected the correctness of the coordinates (when available), we observed several cases for which the coordinates in Wikipedia were inaccurate. The following examples illustrate different reasons for the incorrectness of coordinates in Wikipedia.

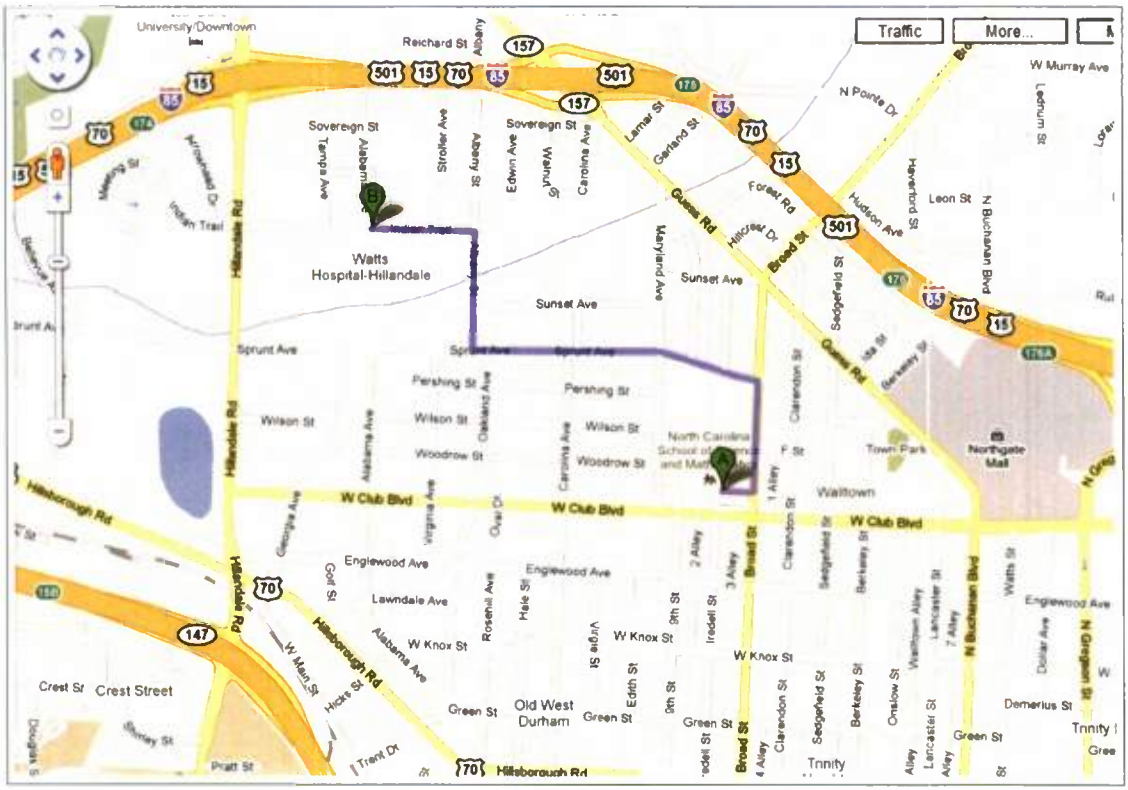


Figure 117: Incorrect Wikipedia coordinate: Coordinate of the parent institution.

1. Coordinate may be of a parent institution. Some hospitals are affiliated with universities. When entering coordinates in a Wikipedia article, the editors may judge the correct reference coordinate differently–that is, the coordinate of a hospital, or that of a parent institution. For example, as shown in Figure 117, the coordinate for "Watts-Hillandale Hospital" in Wikipedia (marker A) represents the location of the parent institution, and therefore, is far away from the correct coordinate (marker B).

2. Coordinate may be of a city. Similar to the above situation, instead of entering the exact coordinate of a hospital, the editors of the article may sometimes simply enter the coordinate of a city. For example, as shown in Figure 118, the coordinate in the Wikipedia article (marker A) for "Carillion Roanoke Memorial Hospital" is at the center of the city, which is far away from the correct coordinate of the hospital (marker B).

3. Coordinate may be of a street with a similar name. Sometimes there may be a street near a hospital, so that their names may be similar. In such situations, the editors of the article may make the mistake of entering the coordinates of the street. Consider "Bryce Hospital" for example, as shown in Figure 119, for which the coordinate obtained from Wikipedia (marker A) is wrong; the correct coordinate is shown by marker B.

4. Coordinate may be approximate Sometimes the mistake may be unintended. The editors may not be careful about entering the precise coordinate. For example, as shown in Figure 120, for "St. Mary's Hospital," the coordinate obtained from Wikipedia (marker A) is about 5 blocks away from the correct coordinate (marker B).

To rigorously quantify the precision of coordinates extracted from Wikipedia, we sampled some records, and manually judged the correctness of their coordinates. For this sampling, rather than using random sampling, we wanted to first classify the hospital records into different partitions, based on the likelihood of the correctness of these coordinates, and then sample randomly within each of these partitions. Therefore, as our criteria for partitioning, we used the distance between the coordinates obtained from Wikipedia and the coordinate generated

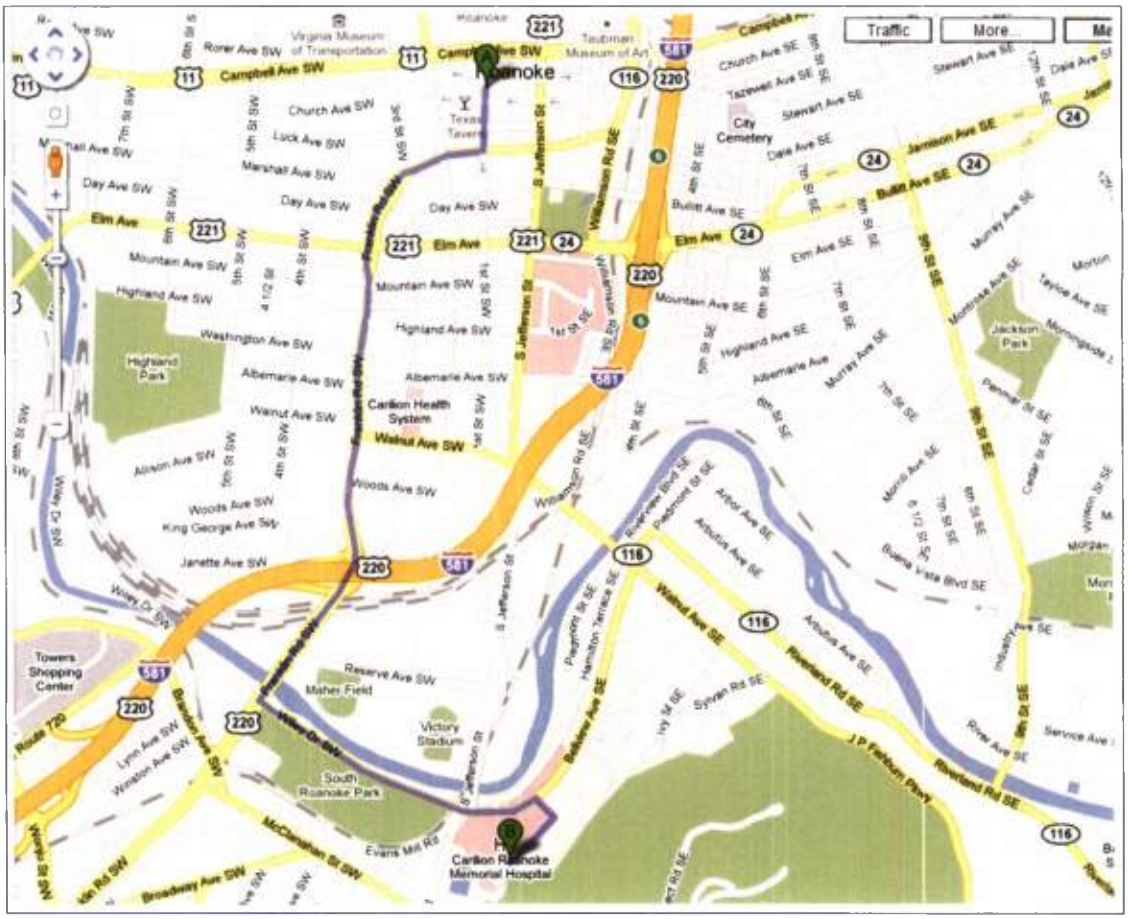


Figure 118: Incorrect Wikipedia coordinate: Coordinate of a city.

by the UsingCity method (introduced later in Section 7.1.2), since the UsingCity method produces coordinates that are the more accurate of the two automatic coordinate generation methods we developed. With this partitioning scheme we expected that the greater the distance between the two coordinates, the more likely that the coordinate from Wikipedia was wrong.

There were 202 cases in which coordinates could be obtained from both methods, *i.e.*, from the Wikipedia article as well as from the UsingCitymethod. Figure 121 shows the distribution of haversine distance between the two coordinates for each of these 202 cases. There were 15 records in which the distance between the two coordinates is zero, *i.e.*, the coordinates were exactly the same. Of these cases, we sampled 9 records, and in all cases found that the coordinates obtained from Wikipedia were correct. For the next partition, there were 172 records for which the distance between two coordinates was between $0 - 0.5$ kilometers; we sampled 10 of these records and found the precision of Wikipedia to be 90%. Moving to the next partition of distance between $0.5 - 1$ kilometers, we found that the precision of Wikipedia drops to 42%. In the last partition of distance–greater than 1 kilometer–the precision of Wikipedia drops to 37.5%. Thus, overall, the precision of the coordinates extracted from Wikipedia was 87%.

7.1.4 Need for Improving the Quality of the Coordinates in Wikipedia

The incompleteness (recall = 34%) and the inaccuracy (precision = 87%) of the coordinates in Wikipedia articles is to be expected. These coordinates are inserted by editors. The editors must manually determine the correct geo-coordinates for the geographic features. Lacking any automated tools to assist them, either the editors do not enter the coordinates (resulting in incompleteness), or the coordinates that they enter are likely to be erroneous

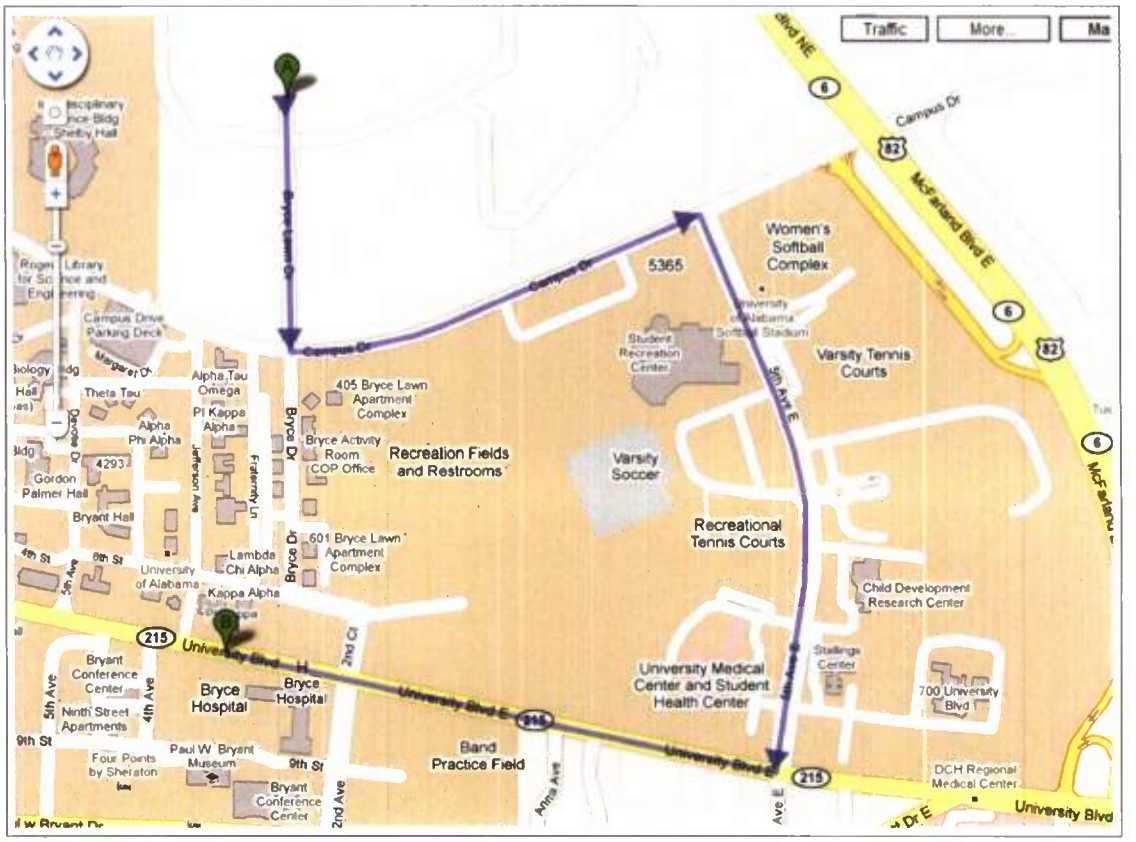


Figure 119: Incorrect Wikipedia coordinate: Coordinate of a street whose name is similar to the name of a hospital.

(resulting in poor accuracy).

However, the limited availability and poor accuracy with regard to coordinates will impact the utility of the resulting gazetteer. Therefore, we explored how to enrich the Wikipedia-driven gazetteer by automatically discovering the correct coordinates for the features found in the Wikipedia articles.

7.1.5 Performance Summary

We summarized the evaluation results of the three methods on the metrics of precision and recall in Figure 122. Both methods that we developed–UsingName and UsingCity– performed better than by obtaining the coordinates directly from Wikipedia. The coordinates obtained directly from Wikipedia articles showed a poor recall result of 34%, and a precision rate of 87%. Our UsingCity method performed better than the coordinates obtained directly from Wikipedia on both recall and precision, respectively, at 49% and 98%. Our UsingName method had a higher recall rate of 78%; however, its precision mark of 93% is lower than UsingCity method.

7.2 *DBPedia* exploration

We identified that a useful application of our technology would be to build a gazetteer comprising of the geographical features present in Wikipedia.

1. The desired target schema would be: (name, FC, lat, lng), with the option of adding other attributes.
2. The target features to capture would include populated places (ADM1, ADM2, towns and villages), landforms (*e.g.*, mountains, lakes, etc.), and buildings and structures (*e.g.*, museums, hospitals, dams, etc.)

Figure 120: Incorrect Wikipedia coordinate: Coordinate is near-by.

Distance	Records	Sampled	Correct	Est. Precision
0	15	9	9	100%
$0 - 0.5$	172	10	9	90%
$0.5 - 1$	7	7	3	42.85%
> 1	8	8	3	37.5%
Total	202			87%

Figure 121: Evaluation of precision of coordinates obtained from Wikipedia.

As the first step of our investigation, we surveyed an existing effort in this area– DBPedia.org. In this section we report our findings from our survey of DBPedia, and how we could provide additional information by overcoming the technical limitation of the DBPedia dataset.

7.2.1 Introduction of DBPedia Dataset

The DBPedia dataset (`http://wiki.dbpedia.org/Datasets`) aims to capture structured information from Wikipedia pages, extracted mostly from the Infobox templates, as illustrated in Figure 123. The dataset is publicly available and comprises of 3.4 million objects. Of these 312,000 represent persons and 413,000 represent geographical features (including 310,000 populated places).

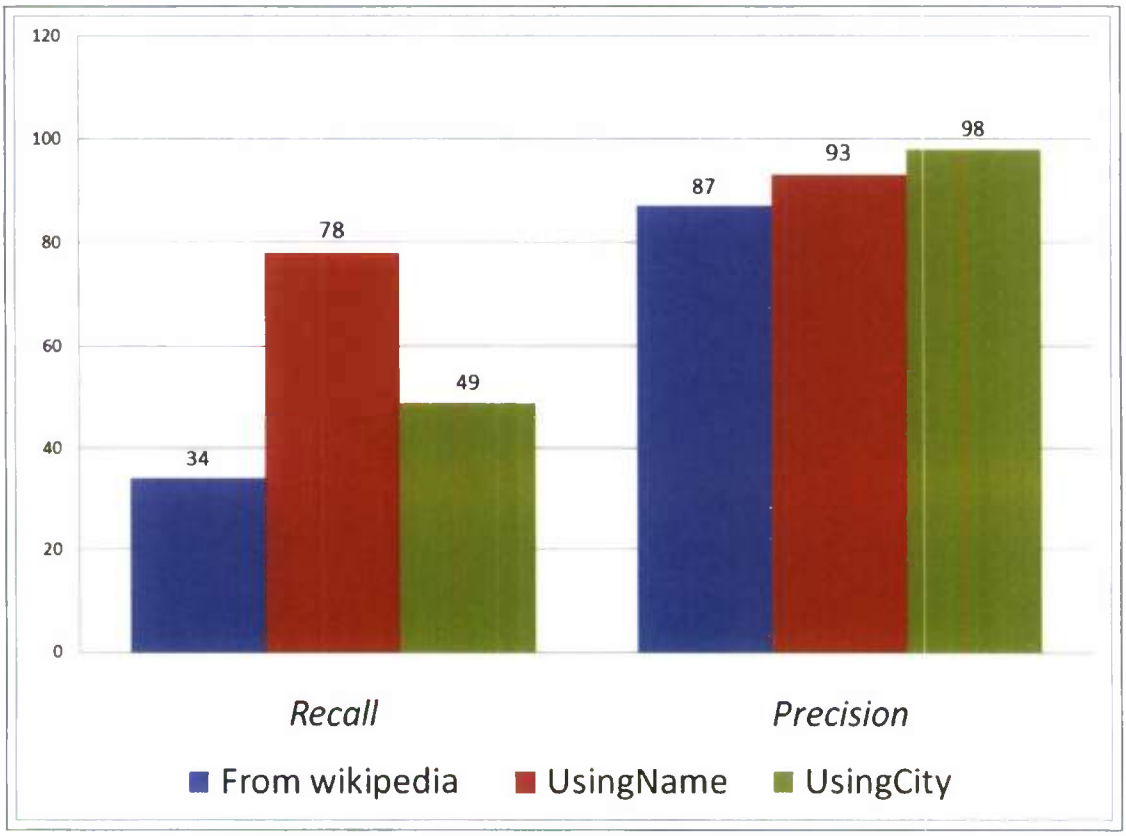


Figure 122: Performance summary: Precision and recall of the coordinates obtained from Wikipedia, the UsingName method and the UsingCity method.

7.2.2 Extraction Technology of DBPedia

Wikipedia has defined various Infobox templates for different categories of objects. When creating an article, the editor needs to decide which category the page belongs to, and based on that choice, pick the matching infobox template in order to insert structured information relative to the topic. For example, `http://en.wikipedia.org/wiki/Category:Geography_infobox_templates` provides the list of infobox templates which can be used for geographical features.

The details of the technology behind DBPedia can be found in several of their publications and presentation talks. A few notable references:

1. `http://www.cis.upenn.edu/~zives/research/dbpedia.pdf`
2. `http://www.google.com/search?q=Extracting+semantic+relationships+between+wikipedia+categories`
3. `http://www.google.com/search?q=What+have+innsbruck+and+leipzig+in+common`
4. `http://lists.w3.org/Archives/Public/www-archive/2007May/att-0056/WWW2007-DBpedia-Talk.pdf`

Essentially, DBPedia operates in a *page-by-page* mode–extracting data from every Wikipedia article that contains an infobox template. It reads the infobox template, and inserts the feature into a structured database with all the attributes specified in the infobox template. It maps each feature into its ontology map based on the categorization of the infobox.

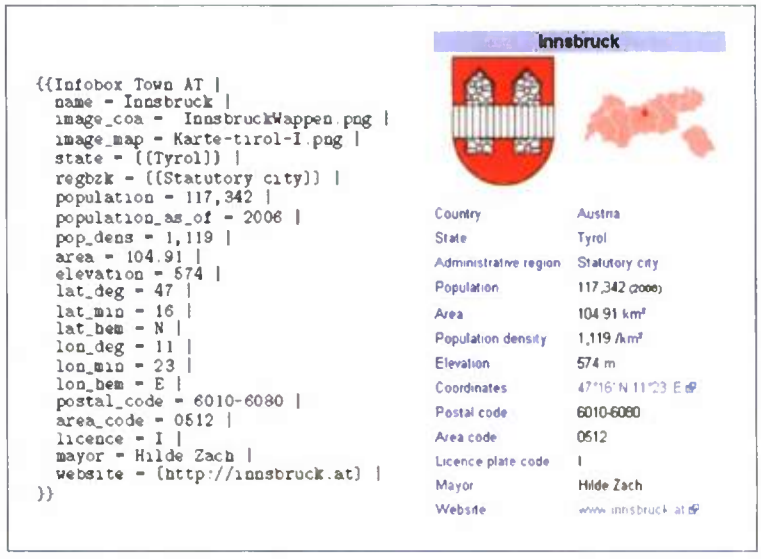


Figure 123: An illustration of an Infobox template used in Wikipedia, from which DBPedia extracts structured information.

7.2.3 Coverage and Quality of DBPedia Dataset

Since the infobox templates already provide information in a structured format, the quality of data in DBPedia can be assumed to be reliable and complete. The users may browse the DBPedia dataset using the search interface at http://dbpedia.neofonie.de/browse. The following links showcase information about some of the features from DBPedia dataset:

1. `http://dbpedia.org/page/Mount_Sinai`
2. `http://dbpedia.org/page/Zabul_Province`
3. `http://dbpedia.org/page/Sela_pri_Zaj%C4%8Djem_Vrhu`
4. `http://dbpedia.org/page/London_Heathrow_Airport`

It appears that the DBPedia dataset may already capture all of our desired attributes for features in which page-by-page operation may be sufficient. Of the four basic attributes in our target schema, three attributes can be found directly from the page for all features, *i.e.*, name, latitude and longitude. The fourth attribute, "FC," can be derived when the template used for the infobox can be used to infer the "FC" attribute at the granularity we desire, *e.g.*, the Wikipedia article for Heathrow Airport uses the infobox template for airports. It seems that this page-by-page extraction approach can derive FC attribute for landforms, and buildings and structures.

7.2.4 Limitation of DBPedia

However, it appears that DBPedia's page-by-page extraction approach cannot determine the correct FC for ADMs. Wikipedia editors use the same infobox template for all the articles about "populated places," and therefore, it is not possible to differentiate the ADM level based on the template used in the infobox.

7.2.5 Overcoming the limitation of DBPedia

We believe that FC attributes for ADM features, which cannot be determined using DBPedia's page-by-page extraction approach, can instead be determined based on the "List" functionality of Wikipedia (more information can be found at `http://en.wikipedia.org/wiki/Wikipedia:Lists`). Wikipedia includes lists for all administrative subdivisions across all countries at `http://en.wikipedia.org/wiki/Category:Administrative_divisions`. We can navigate through these links using our agent crawling technology. During the navigation

process, we can link the features found in the articles at the leaf level, to the parent nodes in the navigation path, and thus, we can correctly infer the FC attribute for ADM features. In particular, we may begin navigation from `http://en.wikipedia.org/wiki/Category:Administrative_divisions`, and continue down to different levels of ADM subdivisions in order to determine the FC feature of a particular geographical features present in the leaf level of the page.

7.3 Alternatives to page-by-page extraction

We believe that the FC attributes, which cannot be determined using the page-by-page extraction approach, can instead be determined for ADM features based on the "List" functionality of Wikipedia (read more at: `http://en.wikipedia.org/wiki/Wikipedia:Lists`). Wikipedia includes lists for all the administrative subdivisions across all countries at `http://en.wikipedia.org/wiki/Category:Administrative_divisions`. We can navigate through the links using our agent crawling technology. During the navigation process, we can link the features found in the articles at the leaf level to the parent nodes in the navigation path, and thus, we can correctly infer the FC attribute for ADM features. In particular, we began navigation from `http://en.wikipedia.org/wiki/Category:Administrative_divisions`, and continued down through the first and second level ADM subdivisions. In doing so we were able to determine the collection of administrative areas associated with each given country in each administrative district level.

Given a particular page, we found it is also possible to traverse back upward through the parent-leaf relationship of *Wikipedia* categories in an attempt to reach a particular broad parent category. We found several uses for upward traversal; given any starting page, it is possible to check if a page is a member of a particular parent category by following each leaf-parent connection until either the category has been reached or a pre-set limit for the height of the leaf-parent tree is passed. If the category is reached, the page exists in the target category; if the pre-set limiting height is always reached, the page does not exist in the target category. During this traversal it is also possible to collect each category traversed as a parent and attach these as attributes to the given starting page. We tested these methods on the collection of ADM areas we found in our parent-leaf, downward traversal of *Wikipedia* categories.

Many *Wikipedia* pages contain templates, a pre-defined structure for displaying certain types of information. In particular, many of the specific *Wikipedia* pages for ADM areas contain a template containing their geocoordinates. We used this structure to attempt to extract latitude and longitude for all pages collected in our downward traversal of *Wikipedia* categories.

Since all *Wikipedia* categories are originally organized by hand (that is, each page must manually be set as existing in a particular category) and could contain errors, we decided that it would be prudent to develop some method of quality analysis to determine the validity of each collection of ADM areas for each country. We also chose to compare our results to the collection of ADM areas gathered by the *database of Global Administrative Areas (GADM)* (`http://www.gadm.org/`).

Finally, we identified one particular area for which information gathered by DBPedia is very scarce. Since *DBPedia* crawls *Wikipedia* pages for its information, all information in *DBPedia* must first exist in *Wikipedia*. That is, if a particular ADM area does not have a page in *Wikipedia*, it will not exist in *DBPedia*. Even so, lists of these areas may exist in *Wikipedia* categories or in pages within these categories, and are specially marked so that the user knows that a *Wikipedia* page does not yet exist for a particular link. Using this information, it is possible to extract additional ADM areas.

7.3.1 Downward traversal of *Wikipedia*

The root of our downward traversal of *Wikipedia* began at the broad level: `http://en.wikipedia.org/wiki/Category:Administrative_divisions`.

Categories in *Wikipedia* can contain sub-categories, pages, or a mix of both. Sub-categories are categories themselves and only differ from a root category by their parent-child association. Pages are direct links to *Wikipedia* pages. We were able to use the titles of both sub-categories and pages to extract the names of ADM areas within a given country. Our general algorithm is as follows:

1. For a given root category:
   (a) Collect a list of interesting sub-categories.
   (b) Collect a list of interesting pages.

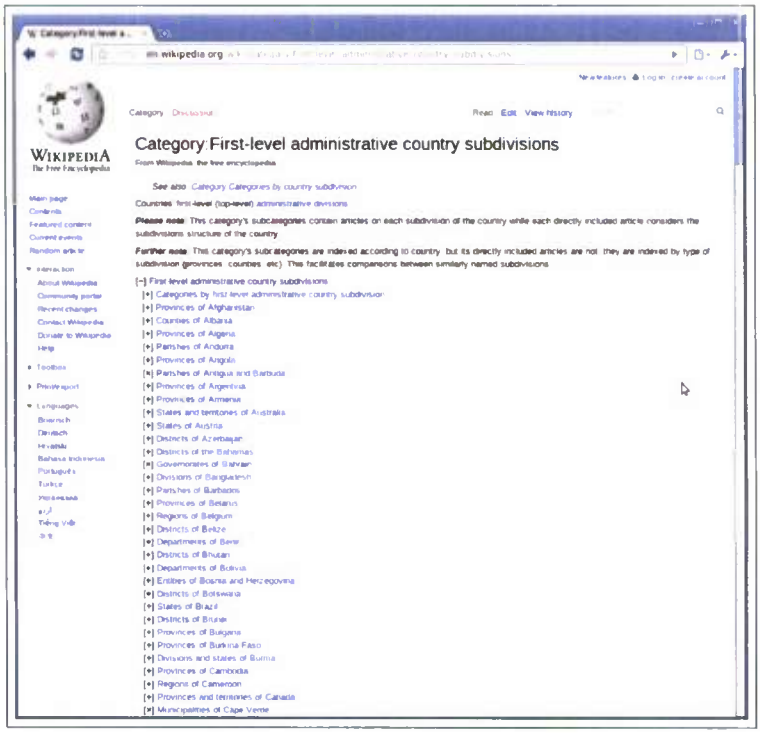


Figure 124: The First-level Division *Wikipedia* category, containing countries as sub-categories.

2. Select a new root category out of the list of collected sub-categories and repeat step 1.
3. End when no more un-traversed, interesting sub-categories exist.
    - Categories that have no interesting sub-categories are marked as leaf categories, since they act as the bottom of the traversal tree.
4. Aggregate lists of interesting pages and leaf categories.

For ease of analysis, we split our inspection of ADM areas into two groups, each a member of our root category:

- `http://en.wikipedia.org/wiki/Category:First-level_administrative_country_subdivisions`
- `http://en.wikipedia.org/wiki/Category:Second-level_administrative_country_subdivisions`

Both contain a similar structure, but are different enough in their internal organization that they require slightly different methods of crawling the downward parent-leaf relationship in order to extract the most-complete set of information.

1: ADM Level 1

The direct children of the First-level Divisions category are sub-categories that represent the first-level ADM divisions of each country. As seen in Figure 124, each of these country-level division categories contain a mix of pages and sub-categories for the specific areas associated with the given country.

We found that, in general, for any given country-level category, the category contains a set of sub-categories and pages that make up the total administrative areas for the region. Using this general fact, we chose to collect a unique set of items from both the sub-categories and pages.

One problem found in gathering sub-categories and pages is that a country-level division might have some non-interesting administrative pages and categories associated with it, in addition to the entries for each ADM area, e.g., a *Wikipedia* page, "`http://en.wikipedia.org/wiki/Provinces_of_Afghanistan`" inside Afghanistan's first-level ADM division that details information about all of Afghanistan's ADM level 1 areas. We solved this problem

by noting that a category's sub-categories and pages are organized into alphabetical blocks. and the non-interesting administrative pages are located outside of the alphabetical ordering.

Another problem is that entries included in the alphabetic ordering are not always valid. Specifically, if a page or sub-category contains the words, "of," "list," "former," or "template," it is not a valid ADM area, but instead a list of some other information associated with the super-category (that is, the country itself). We solved this problem by removing any sub-category or page entries that contained these strings.

Our general rules for inclusion of a sub-category or page as a valid ADM area becomes:

- Item must be included in alphabetical sorting.
- Item must not contain the words "of," "list," "former," or "template".

Our algorithm becomes simply:

1. Descend into the First-level Divisions category.
2. Collect the names of all countries listed as sub-categories.
3. Store these names, along with the associated ADM level 1 type (e.g., "Province", "District", etc...).
4. For each discovered country:
   (a) Descend into the sub-category associated with the given country.
   (b) Collect all sub-categories of the country that fit our rules for inclusion.
   (c) Collect all pages of the country that fit our rules for inclusion.
   (d) Remove duplicate entries from the collection.
   (e) Store all items relationally with their associated country.

Using this method we were able to discover 3850 distinct ADM level 1 areas across 183 distinct countries.

2: ADM Level 2

We found that the general structure of the Second-level Division sub-categories is much the same as the First-level Division sub-categories. We found that the same rules, both alphabet-sorting and the list of strings, were valid for the Second-level Division sub-categories as well as the first, with one major exception. In the First-level Division sub-categories, the unique set of pages and sub-categories (the country's direct children) made a complete list of that country's ADM level 1 areas. In the Second-level Divisions, this proved to be not true.

Zimbabwe, as seen in Figure 125, has several pages associated with its ADM level 2 areas (Districts), which we can directly extract. The sub-categories, however, contain entries like "Districts of Matabeleland North." Descending into this sub-category, we found a mix of pages and sub-categories that included additional ADM level 2 areas associated with Zimbabwe. Since we could not just collect the direct children of a given country to find the complete set of ADM level 2 areas, we had to develop rules for descending into sub-categories.

In surveying several Second-level Division sub-categories, we found that, in general, it would be useful to descend into a country's sub-category if the sub-category contains the word "of." As with the Zimbabwe example, this generally signals that the sub-category will contain a list of additional ADM level 2 areas. One problem we ran into, however, was that sometimes a country contained a sub-category similar to "People of Afghanistan." Using our simple string rule, we would mark this sub-category as one to descend into and collect information from, even though it is clearly not a list of ADM areas. We solved this problem by also making sure that any sub-categories marked for inclusion also contained the string that represented the country's ADM level 2 type (e.g., "District," "County," etc...) Using these rules, we could be reasonably sure that we were only descending into sub-categories that contained additional lists of a given country's ADM level 2 areas.

Our rules for inclusion remained the same as or the First-level Division categories:

- Item must be included in alphabetic sorting.
- Item must not contain the words "of," "list," "former," or "template."

Our rules for descending into sub categories are as follows:

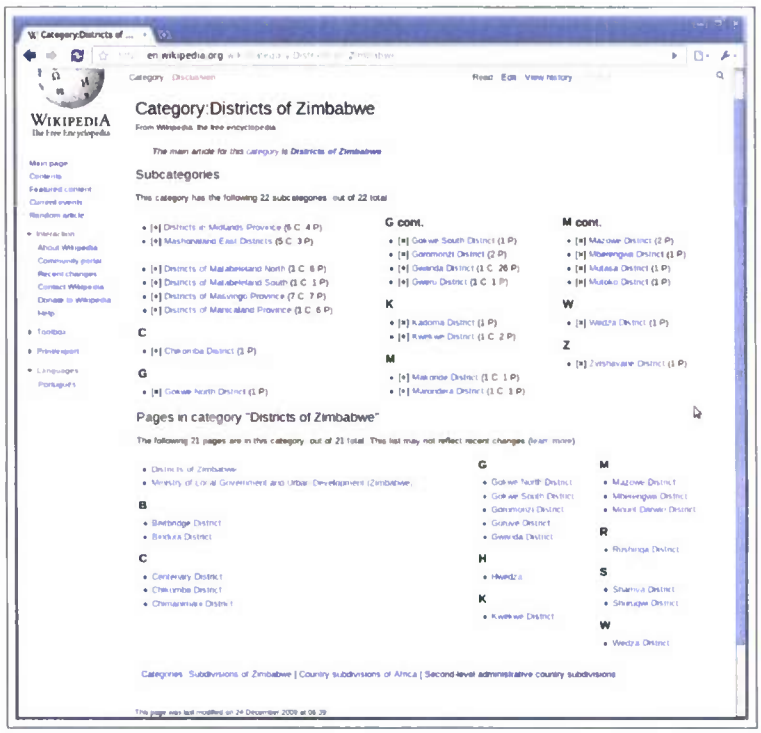


Figure 125: The sub-categories and pages within the Zimbabwe sub-category, in the Second-level Division category.

- Item must include the word "of."
- Item must include the ADM level 2 type in its title.

Our algorithm for collection was a little more complicated than for ADM level 1:

1. Descend into the Second-level Divisions category.
2. Collect the names of all countries listed as sub-categories.
3. Store these names, along with the associated ADM level 2 type (e.g., "Province", "District", etc...).
4. For each discovered country.
   (a) Descend into the sub-category associated with the given country
   (b) Collect all sub-categories of the country that fit our rules for inclusion.
   (c) Collect all pages of the country that fit our rules for inclusion.
   (d) Collect all sub-categories that meet our rules for descending.
      i. Repeat a through d, descending into sub-categories until none matches rules for descending.
   (e) Remove duplicate entries from the collection.
   (f) Store all items relationally with its associated country.

Using this method we were able to discover 18018 distinct ADM level 2 areas across 120 distinct countries.

7.3.2 Upward traversal of *Wikipedia*

Since the parent-child relationship between categories and pages contained within categories is two-directional, that is, just as you can view a list of pages that belong to category, upon viewing a specific page, it will also note that "This page is a category of ...," it is possible to traverse these relationships from the bottom up as well as from the top down.

There are two main utilities associated with crawling *Wikipedia* from the bottom up. The first is that it becomes possible to construct an attribute-list of all categories that a given page belongs to. For instance, given the page "`http://en.wikipedia.org/wiki/Albert_Einstein`", if we traverse its categories upward, we can discover that he belongs to "Violinists," "German Scientists" and "Jewish Pacifists," among many others. By these three categories alone, we can infer much of his heritage, his vocation, political beliefs and hobbies.

The second utility is that, given a specific category to search for, it becomes possible to tell whether or not a given page exists within that category. That is, we can determine that the Bagrami District (`http://en.wikipedia.org/wiki/Bagrami_District`) is an ADM level 2 area of Afghanistan because the Bagrami District page is an entry in the category of `http://en.wikipedia.org/wiki/Category:Districts_of_Kabul_Province`, which is in turn belongs to the category, `http://en.wikipedia.org/wiki/Category:Districts_of_Afghanistan`. This, finally, is an entry in `http://en.wikipedia.org/wiki/Category:Second-level_administrative_country_subdivisions`, which we know lists ADM level 2 areas. The only constraint is that due to the connectivity of Wikipedia categories, it is possible to mis-classify a page if we are allowed to follow too many categories "upward." For example, the page "`http://en.wikipedia.org/wiki/Dennis_Cole`" will eventually end up in the category `http://en.wikipedia.org/wiki/Category:First-level_administrative_country_subdivisions` if allowed to traverse more than 10 levels "upward."

As we can see with the Bagrami District page, generally a page will be linked reasonably closely with the desired category if it actually belongs within the category (within three levels, in this case). Therefore, we can give our algorithm a maximum "height" with which to traverse *Wikipedia*'s categories. If the algorithm fails to find the target category within the height, it is reasonable to assume that the page does not exist within the target category, so long as the height is reasonably small.

We also found that several categories loop through one another, that is, category A exists as an entry in category B, which in turn exists in category C. If category C happens to be an entry within category A, a cycle exists in the tree of category relationships. In order to detect these cycles and avoid an infinite loop, we keep a list of visited categories and do not re-visit categories. Sometimes a specific category will exist within two "branches" of our tree, at varying heights. If we visit the category at a very high height, near our maximum, we may be cut off before we reach interesting information. When we see this category at a much lower height, we would not follow it again, even though the lower height indicates that it may be much more relevant than originally thought. In order to address this, we store the height with which we visited each page in the cycle-detection array. If a given category already exists within the cycle-detection array, but our current height is lower than the old height, we will traverse the category anyway (and update the entry with the new, lower height).

We developed a recursive algorithm to traverse the tree created by *Wikipedia*'s categories. The algorithm's recursive branches will traverse until the category under examination matches one of these base conditions, in which case they will return the specified truth value.

- If the given category has already been traversed at a lower height, return False.
- If the current height is greater than the maximum height, return False.
- If the given category is a member of no additional categories, return False.
- If the given category matches the target category, return True.

Our algorithm is as follows:

1. Initialize an empty array for cycle detection, A.
2. Set target category C and maximum height, H.
3. Set the current page.
4. Examine entry for base conditions against A, C and H.

	Avg pre-error Wiki areas	Avg post-error Wiki areas	Average GADM areas
ADM Level 1	20.58	18.38	14.61
ADM Level 2	139.67	121.73	249.37

Figure 126: The average number of ADM level 1 and 2 areas from pre- and post-error processing Wikipedia, as well as GADM.

Difference in Collected Areas	Number of Occurrences
0	60
<= 10	88
11 – 100	13
101 – 1000	2

Figure 127: A distribution of ADM level 1 differences between *Wikipedia* and the *GADM* dataset, before error elimination.

5. If none of the base conditions match, add current category to array A with the current height. Collect all categories that given page is a member of.

6. Repeat steps 3-5 until no further pages are collected

7. Return logical "or" of all recursive branches. That is, return False if and only if all recursive branches return False. If even one branch returns True, return True.

We used our algorithm to check against a subset of the results returned by our earlier downward traversal of *Wikipedia*. For all 362 ADM level 2 areas we were able to successfully recurse to the category `http://en.wikipedia.org/wiki/Category:Second-level_administrative_country_subdivisions` within two levels.

7.3.3 Quality Analysis

Since members of each category must be included manually, we postulated that it might be possible for errors to exist within the category dataset. That is, a given *Wikipedia* page might be mis-categorized, specifically either a valid ADM area included in the wrong country or a page representing an invalid ADM area (a person associated with a particular country, for instance) that belongs directly to a First-level or Second-level Division category.

We postulated that we might find these errors based on two attributes: if an entry contains a geo-location, that is, a set of latitude and longitude points, it is probably a valid area. If the entry's title contains the type of ADM area that it is supposed to represent (e.g., "Cerrillos Department" of Argentina, since Argentina's ADM 2 areas are called departments), it is also probably a valid ADM area. If we examine articles that do not contain either a geo-location or the ADM area type, we've probably found a page that his been mis-categorized.

For example, if we examine the ADM level 2 areas of Argentina in this manner, we find that 3 of 218 entries fit our rules for exclusion. They are "Partidos of Buenos Aires Province," "Papagayos," and "Punta Delgada." Of these, the first is a category of Partidos in Buenos Aires, the second is a small village in Argentina and the third seems to be some form of tourist area. None of these appear to be valid ADM level 2 areas and thus our rules for exclusion hold true in this area.

Of the 3850 previously-discovered ADM level 1 areas, 412 were marked as invalid, leaving 3438 distinct areas. Of the 18018 previously-discovered ADM level 2 areas, 2314 were marked as invalid, leaving 15704 distinct areas.

We also wanted to examine our dataset against a known source. We chose to compare our results against the data collected in the database of Global Administrative Areas (GADM). We extracted ADM level 1 and 2 areas from the *Google Earth* KMZ maps–similar to our methods used in May, except that we did not care about the specific boundaries, only the names. We were able to collect ADM level 1 information for 163 of 183 countries and ADM level 2 information for 101 of 120 countries.

Figure 126 shows a table containing general information on the two ADM area result sets. Figures 127 and 128 show a binning of the differences between our findings with *Wikipedia* and *GADM*. Figures 129 and 130 show a binning of the differences once we ran our dataset through the previously described error analysis.

Difference in Collected Areas	Number of Occurrences
0	7
<= 10	26
11 – 100	43
101 – 1000	20
1001 – 10000	5

Figure 128: A distribution of ADM level 2 differences between *Wikipedia* and the *GADM* dataset, before error elimination.

Difference in Collected Areas	Number of Occurrences
0	57
<= 10	92
11 – 100	13
101 – 1000	1

Figure 129: A distribution of ADM level 1 differences between *Wikipedia* and the *GADM* dataset, after error elimination.

7.3.4 Non-Existent Wikipedia Pages

We investigated the use of non-existent *Wikipedia* pages to gather further ADM level 1 and level 2 data. Since *DBPedia* can only crawl *Wikipedia* pages once they exist, *DBPedia* misses out on lists of *Wikipedia* links that point to non-existent pages. That is, if a particular ADM level 1 page lists all associated ADM areas with links to their respective pages, a red-colored *Wikipedia* link informs the user that, while the listed item should be a valid page, no one has created an entry for it. Figure 131 displays these links for the Coast Province. Using these lists, it may be possible to extract additional ADM areas from *Wikipedia*'s pages.

The difficulty lies in the fact that *Wikipedia* is only pseudo-organized. Here are three examples of pages that contain non-existent links:

1. `http://en.wikipedia.org/wiki/Ghor_Province`
2. `http://en.wikipedia.org/wiki/Debub_Region`
3. `http://en.wikipedia.org/wiki/Coast_Province`

Although it is very easy to differentiate links to valid pages from links to invalid pages, in examining just these three links we find several major problems that make it very difficult to programmatically determine which links are valid links to process.

We notice that the first page contains non-existent links within the subtext of the article, completely unrelated to ADM information. The second page contains useful non-existent links that point to ungathered ADM information in a table format. Since the page's non-existent links contain only ADM areas, this page could be potentially useful. The third contains ADM information but also nonexistent links to capital cities within the same table structure, with no general or extensible way to tell the difference.

Although there are clearly pages that contain useful, ungathered ADM area information, we can see that it would be very difficult to programmatically discern the difference between useful pages, such as the second, and useless or difficult to parse pages, such as the first and third.

7.4 Feature Gazetteer generation

Following the relative success of our Administrative Area gazetteer generation, we chose to use our "downward traversal" of *Wikipedia* algorithm to generate three additional partial feature-set gazetteers. Using *Wikipedia*'s categorical organization of pages, we collected a ten-country gazetteer for "Populated Places," "Mountains" and "Lakes".

Difference in Collected Areas	Number of Occurrences
0	6
$<= 10$	29
$11 - 100$	39
$101 - 1000$	21
$1001 - 10000$	6

Figure 130: A distribution of ADM level 2 differences between *Wikipedia* and the *GADM* dataset, after error elimination.

Figure 131: An example, in red, of how *Wikipedia* lists links to non-existent pages.

In addition, we chose to include two generic attributes, the latitude/longitude point and *Wikipedia*'s reported textual location (country) and one gazetteer specific attribute, population for populated places, elevation for mountains and depth for lakes.

Finally, we examined our gazetteers against the ontology map provided by *DBPedia* in order to measure the coverage provided by *DBPedia* versus *Wikipedia*'s categories.

1: Populated Places

We located the root category *Populated Places by Country* (`http://en.wikipedia.org/wiki/Category:Populated_places_by_country`) and selected 10 reasonably populated country sub-categories for traversal. Similar to our prior downward traversal, we chose to aggregate the list of *pages* and *sub-categories* within each country's category. We used the same rules for inclusion as before, including sub-categories that existed within alphabetical ordering and excluding pages and sub-categories that contained certain words (*e.g.* "list," "template").

Figure 132 gives a short summary of our findings. In addition, we were able to extract latitude/longitude points for 75.34% of the features listed in our gazetteer. We found population information for 54.30% and *Wikipedia* reported the country for 87.15% of the extracted features.

Further, when checked against *DBPedia*'s ontology map we found that *DBPedia* was missing ontology mappings entirely for 26.31% of features. We believe that this is because many of the extracted pages did not contain an

Country	Gazetteer size	NGA Gazetteer
Afghanistan	822	31045
Colombia	1052	26341
Samoa	156	400
Serbia	931	8430
Suriname	113	415
Sweden	2130	24762
Tajikistan	305	2397
Tunisia	283	1722
Uganda	235	5483
Zimbabwe	259	1386

Figure 132: A per-country breakdown of the number of populated places acquired for each country. Also features the number of populated places featured in the NGA gazetteer.

Country	Gazetteer size	NGA Gazetteer
Afghanistan	14	13869
Brazil	27	706
France	66	1062
Indonesia	147	13565
Kosovo	65	0
Mexico	31	5377
New Zealand	84	3241
Switzerland	912	265
Turkey	41	4060
Venezuela	23	4092

Figure 133: A per-country breakdown of the number of mountains acquired for each country.

infobox, which *DBPedia* relies on for ontology mappings.

2: Mountains

We located the root category *Mountains by Country* (`http://en.wikipedia.org/wiki/Category:Mountains_by_country`) and selected 10 reasonably populated country sub-categories for traversal. We again used the same rules as from "Populated Places" and as in our prior "downward" traversal of *Wikipedia*.

Figure 133 gives a brief summary of our findings. In addition, we were able to extract latitude/longitude points for 92.62% of our mountain features. We also extracted the elevation for 86.67% and *Wikipedia* reported the location for 48.58% of features.

Further, when checked against *DBPedia*'s ontology map we found that *DBPedia* was missing ontology mappings entirely for 14.61% of features. Again, we believe that this is because many of the extracted pages did not contain an infobox, which *DBPedia* relies on for ontology mappings.

3: Lakes

We located the root category *Lakes by Country* (`http://en.wikipedia.org/wiki/Category:Lakes_by_country`) and selected 10 reasonably populated country sub-categories for traversal. We again used the same rules as from "Populated Places," "Mountains" and our prior "downward" traversal.

Figure 134 gives a brief summary of our findings. In addition, we were able to extract latitude/longitude points for 74.95% of our lake features. We also extracted the depth for 26.19% and *Wikipedia* reported the location for 92.96% of features.

Further, when checked against *DBPedia*'s ontology map we found that *DBPedia* was missing ontology mappings entirely for 23.29% of features. Again, we believe that this is because many of the extracted pages did not contain an infobox, which *DBPedia* relies on for ontology mappings.

Country	Gazetteer size	NGA Gazetteer
Bolivia	108	203
Chile	58	435
China	58	644
Estonia	162	246
Hungary	20	285
Italy	66	161
New Zealand	174	345
Russia	71	13717
Switzerland	217	66
Turkey	32	212

Figure 134: A per-country breakdown of the number of lakes acquired for each country.

7.5 *Wikipedia* autocorrection

The linked gazetteer which has been generated for the *Afghanistan Places Profile Capstone* project as described in Section 8 is well suited to perform tasks which require well-linked feature data, such as the location of, and addition to, missing attributes within *Wikipedia*. Since *Wikipedia* uses well-structured attribute-data in their *Infobox Templates*, the attributes which have been aggregated for the *Capstone* system might be used to "fill-in" missing *Infobox* data.

In this task, we have built a *Wikipedia Correction Interface* from which an operator may add or correct *Wikipedia Infobox Template* information.

7.5.1 Problems in *Wikipedia* Articles

Wikipedia faces several challenges with the organization of their articles. These can be attributed to the vast number of articles, imperfect automated editing and simple human error, and fall into two larger categories:

- **Mis-Categorization:** *Infobox Templates* and even entire articles might be mis-categorized. *e.g.*. "Papagayos" (`http://en.wikipedia.org/wiki/Papagayos`) is categorized as a "Department of Argentina," but is actually a small village in the *Chacabuco* Department.
- **Missing Attribute Information:** We found that only 54.3% of extracted Populated Place features had a "population" attribute.

Since *Wikipedia* articles which are linked in our concordance mapping should be de facto correctly categorized, the *Afghanistan Places Profile* data is not well suited to discovering instances of mis-categorization. The concordance dataset is, however, very good at filling-in missing attribute information, since *Wikipedia* articles have been linked explicitly to populated place features from several other data-sets.

It is for this reason that we developed a console upon which *Wikipedia* articles can be examined for missing *Infobox* attribute information and, when such data is available in our concordance data-set, can be corrected.

7.5.2 *Wikipedia Infobox* Structure.

In order to extract *Wikipedia* infobox data, we used their publicly available *MediaWiki API* [29]. This API is provided by the *Wikimedia Foundation* and allows direct access to portions of *Wikipedia* articles. Although the API allows a user to query specific sections of an article, which are usually preceded by a textual heading and a horizontal, ruled line, it does not allow for the direct extraction of an *Infobox* template.

Infobox Templates are placed in the "0^{th}" section of a *Wikipedia* article, generally right before the article's introductory text. This can be seen in Figure 135, where we note that the *Infobox* is placed in *Ab Gaj*'s introductory section, which begins, "Ab Gaj (or Abgach) is a village in Badakhshan Province in north-eastern Afghanistan. ..." In the lower portion of this figure, we see the raw text of the 0^{th} section of this *Infobox Template*. Here we can again see that, though it begins with the *Infobox*, denoted by the yellow highlighting, it also contains additional text.

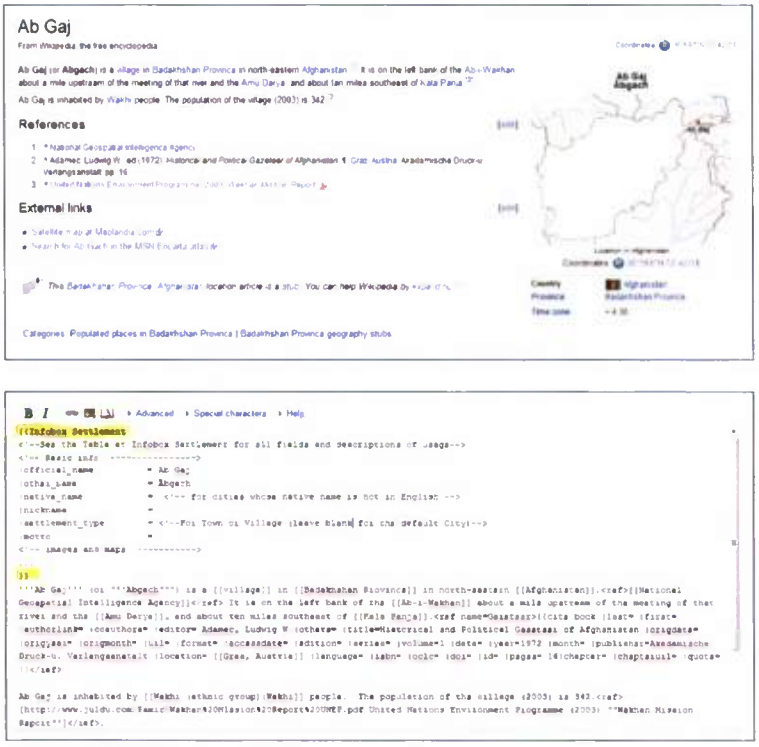


Figure 135: The *Infobox Template* for Ab Gaj, presented graphically (left) and as raw text (right).

Because of this, before we can edit the *Infobox* directly, we must first parse it out from the rest of the article's text.

7.5.3 Extracting *Wikipedia Infobox Templates*

We know that the *Infobox* will always start with the string, "Infobox settlement" in the case of populated places. Logically, we know that the template's end is denoted by a set of closing brackets (""). With this knowledge, we can begin our parsing of the template when we find that beginning string. We cannot, however, end parsing upon reaching the first set of closing brackets that we come across. The *Wikipedia Infobox* syntax allows for sub-templates to be inserted within the larger "parent" template. That is, we might have an infobox that looks as such:

```
{{Infobox settlement
<!--See the Table at Infobox settlement for all fields and descriptions of usage-->
<!-- Basic info  ----------------->
|name                    = Kabul
|settlement_type         = City
...
|coordinates_region      = AF
|subdivision_type        = Country
|subdivision_name        = {{Flag|Afghanistan}}
...
}}
```

Here we see that the attribute "subdivision_name" contains a sub-section which is delimited by double-brackets. Because of this, we must recognize when a new sub-section is instantiated so that we do not accidentally mistake its closing brackets for the end of the *Infobox*. To do this, while iterating over the template, we keep a count of

the number of open double brackets ("{{") and closed double brackets ("}}"). When these numbers are equal, we know that we have reached the closing brackets which indicate the end of our *Infobox Template*.

For example, before iterating over the *Kabul* infobox, we initialize two variables, *open_count* and *closed_count* and set them to zero. Upon reaching the opening line "{{Infobox settlement" we increment the *open_count* so that its value is now 1. Since the *closed_count* is zero, we continue iterating over the infobox. Upon reaching the "subdivision_name" line, we encounter the first open double brackets, "{{Flag ..." We once again increment the *open_count* so that its value is now 2. Immediately after, we encounter the closing brackets for this sub-section, "Afghanistan}}." We now increment the *closed_count* so that its value is 1, and since $open\_count \neq closed\_count$, we continue iterating. Finally, we reach the last line of the *Infobox*, "}}." We once again increment the *closed_count* so that its value is 2. Since $open\_count = closed\_count$ we know that we have reached the end of the *Infobox* and can safely extract this text for further parsing.

After extracting the *Infobox* section, we also save any preceding and succeeding text so that we can re-insert it once the attribute-addition is completed.

7.5.4 Parsing *Wikipedia Infobox Templates*

Now that the *Infobox* has been successfully extracted, it must be parsed so that existing attributes can be labeled and missing attributes can be added. In the template syntax, a "|" character denotes the beginning of a new attribute. Therefore, we first mark all of the locations of the "|" character. One note is that we do not mark the location of the "|" when it is inside a sub-section, denoted by interior open and closed double brackets. In this instance, we are only concerned with editing the values of primary attributes, not values contained within sub-template sections. We also do not mark the "|" character when it exists within HTML comments, which are denoted by the opening sequence, "<!-" and closing sequence "->." These areas will not be displayed when the template is rendered and often contain explanatory notes.

Once all of these locations are noted, we split the *Infobox* into chunks based on these positions. This gives us sets of attribute-value pairs which we can be further parsed. Finally, we split each pair into two sub-objects based on the location of the "=" sign. We now have a pointer to each key (the section before the "=") and its value (the text after the "="). After this step is completed, we now have a fully-formed dictionary of the key-value pairs within the *Infobox*.

The *Wikipedia* bot is now ready to take user-input. Upon receiving additional attribute-values, our system prepares text for re-insertion.

7.5.5 Preparing *Infobox* for re-insertion within its *Wikipedia* article

Before re-insertion of the template can occur, we must first solve several problems:

- Wikipedia infobox templates have a specific syntax with known set of attributes. Since not all linked pages have infoboxes, and since the Wikipedia Bot should place new attributes in order, it is important to know the correct infobox syntax.
- Several attribute fields can have multiple entries, e.g., "subdivision_name" can have many numbered entries: "subdivision_name", "subdivision_name1", *etc.*. These instances correspond to different levels of administrative divisions within a country.
- Since there is no way to know if the known set of attributes is complete, we must preserve attribute-value pairs found within a given page that are not found in the known list. That is, we may only have the entry for "subdivision_name" in the known-syntax for populated places, but if additional "subdivision_name" levels have been defined, we must re-insert them – and re-insert them in their proper order.

Our algorithm for preparing the *Infobox Template* for re-insertion is as follows:

1. Initialize a list of "known attributes"
2. Initialize an array of user-supplied corrected attributes
3. Run any formatting rules on user-supplied attributes
    - Latitude and longitude are split into degrees/minutes/seconds as separate fields within Wikipedia, but are retrieved as one field from the user.

4. Initialize an array of extracted attributes
5. Initialize an array for the corrected *Infobox*
6. For each item within "known attributes":
   (a) Check to see if the attribute exists within the user-supplied array. If it does, take its value and place the attribute-value pair in the corrected attributes list
   (b) Else, check the extracted attributes for a value, if it exists, append the attribute-value pair to the corrected attributes list
   (c) Else, append the attribute with a blank value
   (d) Once appended, check the extracted attributes array for the attribute's next sibling. If it does not exist within the known attributes array, take its attribute name and value and append to the corrected attributes list. Continue until a known attribute is discovered; once a known attribute is discovered, end appending and continue known-attribute iteration as before. This ensures that any additional attributes which may not exist in the known attributes array are saved and re-inserted into the *Infobox*.

For example, consider the *Infobox Template* for the city of Ghorak:

```
{{Infobox Settlement
|official_name = Ghorak
|subdivision_type = Country
|subdivision_type1 = [[Provinces of Afghanistan]]
|timezone = [[UTC+4:30]]
}}
```

Our "known attributes" are "official_name," "subdivision_type" and "timezone." While iterating through the "known attributes", we check ahead in the extracted attributes to see if the next attribute exists in the known list. If it does not, we will append it to our re-insertion text and continue looking ahead until we find an attribute that exists within the known list.

- Known Attributes: official_name, subdivision_type, timezone
- Extracted Attributes: official_name, subdivision_type, subdivision_type1, timezone

1. Check extracted attributes for "official name", it exists so we pull its value, check its next sibling "subdivision_type". This exists within our known list, so we do nothing.
2. Check for "subdivision_type", it exists so we pull its value. Its next sibling, "subdivision_type1" does not exist in the known list, so we append it after "subdivision_type". We then check "subdivision_type1" for its next sibling, which is "timezone". Since "timezone" exists, we stop appending and continue iterating over the known list.
3. Check for "timezone", it exists so we pull its value. There are no additional attributes in the known list or the extracted attributes, iteration ends.

7.5.6 Re-insertion of the *Infobox Template*

Once this corrected list is created, the *Infobox* can be re-built and inserted back into *Wikipedia*. First, we initialize a string for the *Infobox* which starts with the proper syntax, "{{Infobox settlement." Next, we iterate over our corrected values array and append each key-value pair, one per line, to this string. Finally, we append the closing double brackets, "}}" which signal the end of the *Infobox*. We then re-insert any preceding and succeeding text which was gathered during the *Infobox* extraction.

We then take our finished string and re-insert it back into the 0^{th} section of the *Wikipedia* article. This process ensures that new information can be added to the *Wikipedia* article while preserving all data which existed there before.

The user must also provide their own login information to the *Wikipedia* bot. This is to ensure that the user has carefully reviewed their editing decisions and gives the *Wikipedia* editors a marker for double-checking the

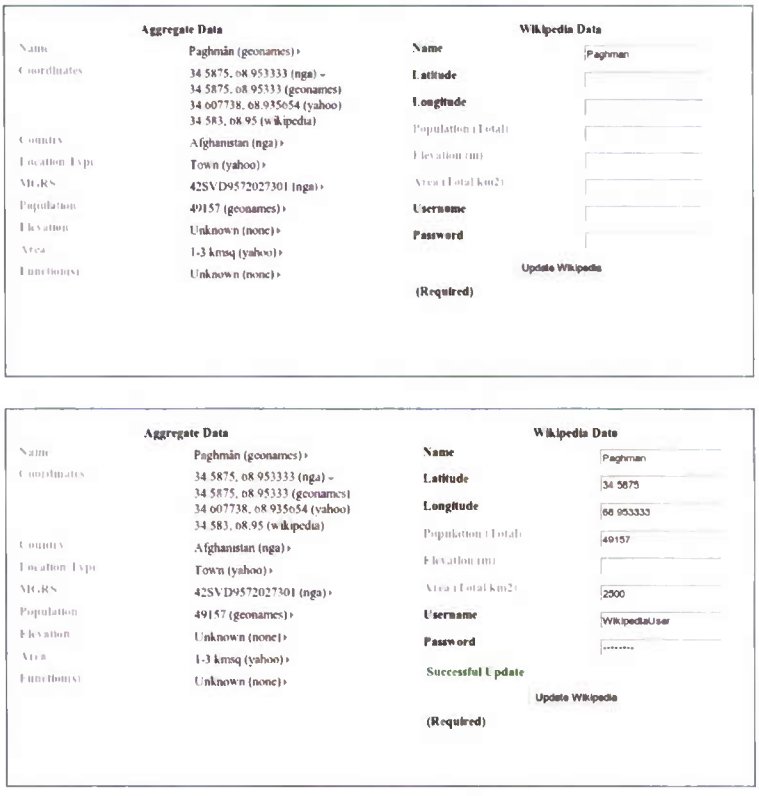


Figure 136: A before (top) and after (bottom) view of the *Wikipedia Correction Bot* acting on the *Infobox Template* for Paghman.

newly added attribute information. Unfortunately the Wikipedia API does not support 3rd party authentication protocols such as OAuth, so login information must be sent in plain-text. This might be a slight security concern because if someone performs a man-in-the-middle attack, they might be able to extract a user's *Wikipedia* login information. There are some plans to implement a more secure login system for the API in the future [18], which can then be implemented in the *Wikipedia Correction Bot*.

Finally, sometimes a captcha prevents the bot from editing a particular Wikipedia page. In these events, our bot recognizes the need for a captcha and forwards the image back to the user. The user then enters the captcha text and resubmits their changes. This is to ensure that no automated process can negatively effect a large subset of *Wikipedia* articles without human oversight.

7.5.7 *Wikipedia* Correction Bot Conclusions and screen-shots

The Wikipedia Correction Bot allows users to fill in missing attribute information with known good values. The Wikipedia Correction Bot operates first and foremost on the principle "do no harm," that is, the Wikipedia Bot should never programmatically delete information which resides within a Wikipedia Article.

We have also built a basic front-end console for operation of the bot. Figure 136 shows a "before" and "after" view of the editing console acting on the *Wikipedia* article for Paghman. Note that in each picture, the attribute data as aggregated by the *Capstone* system resides on the left, while the attributes captured by *Wikipedia* are on the right.

8 GeoEngine Capstone System

Technical Capabilities

Throughout the *GeoEngine* project, we have explored various forms of geospatial data generation and augmentation for a number of different geospatial features from a wide variety of sources. In these tasks, we have developed many techniques for extracting, aggregating and categorizing different forms of geospatial data:

1. Missing Attribute Discovery and Correction: We are able to add missing information to existing gazetteers such as population for populated places within Benin and Germany and elevation for mountains within Afghanistan.

2. Feature Generation: We are able to discover new features within a target geography such as hospitals in Chicago and Mosques within Afghanistan.

3. Extraction from Multiple Sources: In order to increase the robustness of our feature data we are able to extract information from a variety of sources.

4. Conflict Resolution: We are able to rank various candidate answers in order of the confidence of their correctness.

5. Geospatial Disambiguation: We are able to resolve ambiguities which arise from merging geospatial features from disparate sources, including conflicting names for a given place or a given name referring to multiple places.

6. Operation Consoles: We are able to develop intuitive operator consoles in order to provide non-technical interfaces to the aggregate information collected by our various tasks.

Capstone Project

While developing these techniques, we have discovered an additional problem area: that of the stratification of attribute-information associated with various geospatial features across many disparate datasets. We believe that we can apply techniques developed within our previous projects to identify and solve the sub-problems associated with presenting unified attribution for a given geospatial feature.

There exist a number of geospatial gazetteers which display attribute information for various geospatial feature sets *e.g.*, a "mountain" gazetteer which might contain geospatial coordinates and elevation or a "lake" gazetteer which might contain the feature's surrounding countries and its average depth. Further, for any one of these geospatial features, there exist a number of disparate gazetteers which contain both overlapping and non-overlapping attribute information for a given feature. These gazetteers are generally released from different organizations such as the *National Geospatial-Intelligence Agency* (*NGA*) or *GeoNames* and are organized and indexed separately, in an overall non-connected, non-unified structure.

Specifically, this problem is especially prevalent within the "populated place" geospatial feature. One gazetteer might contain population information, while another holds the feature's elevation. One gazetteer might list only the feature's primary name, while another will list all of the feature's secondary names. Even when two gazetteers share an attribute, one gazetteer may populate the attribute more thoroughly than another. For example, *GeoNames* lists population of the city of Kabul, Afghaninstan at $3,043,532$, but that figure is completely absent from the *NGA* dataset.

We find this problem area to be particularly relevant to the *GeoEngine* project and the United States Army. Consider a soldier deployed in a foreign locality with little ground intelligence, or an intelligence operator who would like access to a comprehensive profile of a certain locality, including its basic properties, relationship to other geographical features, key amenities or sub-city features including hospitals, banks or postal offices.

We've identified a number of types of sources which contain various types of information and classified them into different categories:

- Properties: Properties associated with the property features of a populated place, *e.g.*, elevation, coordinates, population and name. Sources include the *National Geospatial-Intelligence Agency* (*NGA*), *GeoNames*, *Yahoo GeoPlanet*, *Wikipedia* and *Locode*.

- Sub-City Features: Features that exist within a populated place, *e.g.*, mosques, schools and hospitals. Sources include the *Open Street Map* (*OSM*) project and *Wikipedia*.

- Real-Time Information: Transient information that must be gathered in real time, *e.g.*, the weather or local news. Sources include *Yahoo! Weather* and *Al Jazeera News*.

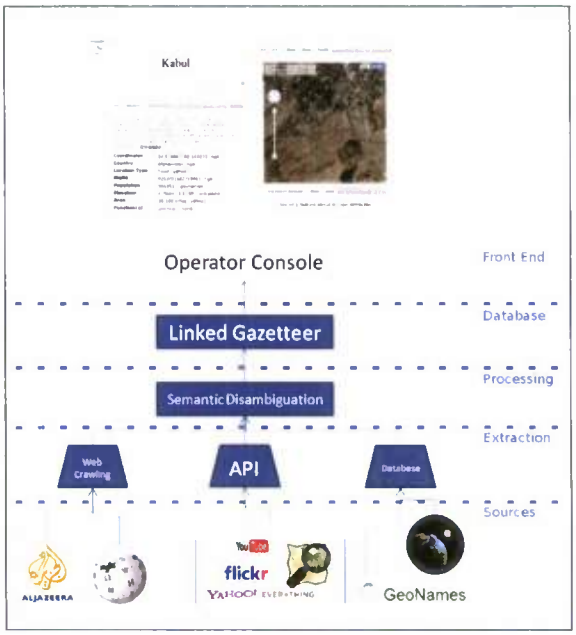


Figure 137: The planned multi-layer architecture of the Places Profile.

- Multimedia Information: Audio and Visual information associated with a populated place. Sources include *Flickr*, *YouTube* and *Picasa*.
- People and Events: People and events which are associated with a populated place. Sources include *Wikipedia* and various news sources.

8.1 System design

8.1.1 Places Profile: System Design

Based on our experience in building different technology modules throughout prior tasks, we plan a multi-source, multi-layer architecture for the Places Profile (see Figure 137).

At the base of our system lies the source layer. Information is aggregated from a diverse set of sources, from existing feature gazetteers such as the *NGA* to media items from *Flickr* to community projects such as the *Open Street Map* project.

The information contained within these disparate sources must be extracted in various ways; this extraction layer presents an adaptable array of methods with which to extract data from our sources. Some sites provide a public API from which we can request information, while others allow a user to extract a snapshot of their entire dataset. Still others must be crawled directly from the Web.

Once information is successfully extracted from these sources, we must develop some methods for linking the data. That is, if two datasets contain an entry for "Kabul, Afghanistan," we must have some way to explicitly link these features in order to aggregate the attribute information associated with them in each disparate dataset. The processing layer is responsible for merging feature data, as well as handling conflicts that arise from such unions.

The aggregate information must then be stored in a structure suitable for easy perusal. Since multiple types of information – property, sub-city, real-time, multimedia and event data – will be stored and semantically linked within the Places Profile, a variety of structures must be utilized in order to logically retrieve results from our final dataset. That is, information about the weather in Herat, Afghanistan is held in an intrinsically different structure than the number and location of mosques within the same city.

Finally, our Places Profile would be considerably less useful without a visual operator console from which information can be displayed in a non-technical manner, *i.e.*, without requiring knowledge of specific database

Attribute	Description
UFI (100%)	The unique index assigned to each populated place feature.
LAT (100%)	The latitude portion of the feature's geospatial coordinate, in dotted decimal format.
LONG (100%)	The longitude portion of the feature's geospatial coordinate, in dotted decimal format.
MGRS (100%)	Military Grid Reference System (MGRS) coordinates. MGRS is an alpha-numeric system for expressing UTM/UPS coordinates. A single alpha-numeric value references an area that is unique for the entire Earth.
DSG (100%)	Feature Designation Code. A two to five-character code used to identify the type of Geoname feature a name is applied to.
CC1 (100%)	Primary Country Code. A two character alphabetic code from the Geopolitical Codes (formerly FIPS 10-4 Standard) that uniquely identifies a geopolitical entity (countries, dependencies, and areas of special sovereignty).
ADM1 (100%)	First-order administrative division code. A two character alpha-numeric code from the Geopolitical Codes (formerly FIPS 10-4 Standard) describing a primary administrative division of a geopolitical entity, such as a state in the United States.
POP (< .01%)	Population figures.
FULL_NAME_RO (100%)	Full name - reading order. The full name is the complete name that identifies a named feature. The full name is output in reading order, "Mount Everest", vs. reversed generic, "Everest, Mount", as stored in the database.

Figure 138: A table containing the property attributes collected from the *NGA*.

systems or query languages. This operator console also allows for visual inferences to be made about data within a given place, *e.g.*, the proximity of several mosques may signal a previously undefined religious center.

It is important to note once again that, while the current development model integrates only free, open source datasets, the techniques developed can easily incorporate proprietary or classified sources of data.

8.2 Properties gazetteer

8.2.1 Properties Gazetteer Aggregation

Each gazetteer contains a number of overlapping and non-overlapping attributes. The overlapping attributes are valuable because they allow a means of comparison between each dataset, while the non-overlapping attributes are valuable because they add more pieces of interesting information to our final gazetteer.

We will describe their various attributes and their recall value from each of our chosen primary gazetteers in the following sections.

1: ***National Geospatial-Intelligence Agency (NGA)***

The *NGA* is the United States Department of Defense's primary mapping organization, and is the most feature-complete of our chosen datasets. Their data is publicly available at `http://earth-info.nga.mil/gns/html/` and can be downloaded as a tab-delimited file on a per-country basis or for the entire world. This file contains all named features for a given country. We were able to import the data into a SQL database, then, using the dataset's *Feature Designation Code*, we extracted all *populated place* features. In total, we obtained 32,380 populated place features in Afghanistan identified by a *Unique Feature Identifier*.

The gazeteer's primary attributes are described in Figure 138, with descriptions taken from the *NGA* at `http://earth-info.nga.mil/gns/html/gis_countryfiles.html`. An example of extracted values for three cities can be seen in figure 139. We notice that the *NGA* dataset contains very few interesting attributes and is used primarily as a basis for uniform spelling of geographic names. We discover that both population and elevation

Attribute	Kabul	Kandahar	Herat
UFI	−3378435	−3379064	−3377359
LAT	34.516667	31.613323	34.348167
LONG	69.183333	65.710126	62.199673
MGRS	42SWD1682719461	41RQR5709800766	41SMU2639401051
DSG	PPLC	PPLA	PPLA
CC1	AF	AF	AF
ADM1	13	23	11
POP	0	0	0
FULL_NAME_RO	Kabul	Kandahār	Herāt

Figure 139: A comparison of values gathered from the *NGA* dataset.

Attribute	Description
geonameid (100%)	Integer ID of record in *GeoName*'s database.
name (100%)	Name of the populated place feature.
asciiname (99.98%)	Name of the populated place feature in plain ascii characters.
alternatenames (97.33%)	Alternate names for the populated place feature, comma separated.
latitude (100%)	Latitude in decimal degrees.
longitude (100%)	Longitude in decimal degrees.
country code (100%)	ISO-3166 2-letter country code.
admin1 code (100%)	FIPS code.
population (.23%)	Population figure.

Figure 140: A table containing the property attributes collected from *GeoNames*.

have very low recall and that the records contain administrative area division information only at the broadest resolution.

One interesting thing to note is that entries in the *NGA* are on a per-name basis. That is, when a particular feature (as identified by its *UFI*) has multiple names, there will be multiple entries in the dataset. We use the *NGA*'s "Name Type" field in order to identify the feature's "primary name," then aggregate the rest as a list of "secondary names."

2: *GeoNames*

GeoNames is a "geographical database ... available for download free of charge under a Creative Commons attribution license. It contains over 10 million geographical names and consists of 7.5 million unique features whereof 2.8 million populated places and 5.5 million alternate names." [5] Like the *NGA*, *GeoNames* provides tab-delimited country files for download. After importing to a SQL database, we were able to extract 30,875 features using *GeoName*'s *Feature Code* to identify *populated places*. Features are identified by *GeoName*'s *geonamesid*.

Their primary attributes are described in Figure 140, with descriptions taken from `http://download.geonames.org/export/dump/readme.txt`. An example of extracted values for three cities can be seen in Figure 141. We note that *GeoNames*, similar to *NGA*, provides few additional attributes. Instead, it provides another rich list of primary and secondary names for populated place features, which can be used to supplement the set of names provided by the *NGA* gazetteer.

3: *Yahoo*

Yahoo GeoPlanet is a public service which provides access to geo-tagged features via a REST API. All geographic features have a unique *Where On Earth ID (woeid)* assigned to them. The API allows traversal of a given *woeid*'s descendants – that is, given the *woeid* for Afghanistan, all geographic features contained "within" Afghanistan can be traversed. Using this traversal, we extracted 628 total records which corresponded to *Yahoo*'s *Place Type Name* "Town" as *populated places*.

[5] Taken from the description found at `http://www.geonames.org/about.html`.

Attribute	Kabul	Kandahar	Herat
geonameid	1138958	1138336	1140026
name	Kabul	Kandahār	Herāt
asciiname	Kabul	Kandahar	Herat
alternatenames	Cabool, Caboul, Cabul, ...	Candahar,Kandagar,Kandahar, ...	Gerat,Herat,Herāt, ...
latitude	34.52813	31.61332	34.34817
longitude	69.17233	65.71013	62.19967
country code	AF	AF	AF
admin1 code	13	23	11
population	3,043,532	391,190	272,806

Figure 141: A comparison of values gathered from the *GeoNames* dataset.

Attribute	Description
woeid (100%)	Where On Earth Identifer.
placeTypeName (100%)	Localized name of the place type.
name (100%)	Localized name of the place.
latitude (100%)	Latitude in decimal degrees.
longitude (100%)	Longitude in decimal degrees.
country (100%)	Localized name of the country.
admin1 (100%)	Localized name of the subcountry admin area.
admin2 (100%)	Localized name of the subadmin1 admin area.
locality1 (100%)	Localized name of a populated place such as a town or village.
locality2 (01.59%)	Localized name of the sublocality such as a suburb or neighborhood).
popRank (98.41%)	Binning code representing the population size, *e.g.*, "10" refers to a population of 30,000-100,000.
areaRank (98.41%)	Binning code representing the size of the place, *e.g.*, "10" refers to an area of 30,000-100,000 km^2

Figure 142: A table containing the property attributes collected from *Yahoo*.

Yahoo's feature's primary attributes are described in Figure 142, with descriptions taken from `http://developer.yahoo.com/geo/geoplanet/guide/api-reference.html`. An example of extracted values for three cities can be seen in Figure 143. We notice immediately that *Yahoo* contains a much higher recall for a populated place's population. We also notice a high recall for a new attribute, *Yahoo*'s *areaRank*, which gives an approximation of a populated place's area. We believe that this high recall is attributable to *Yahoo*'s low total recall across populated places within Afghanistan – we were able to extract only 628 total places as opposed to the more than 30,000 obtained from *NGA* and *GeoNames*. We believe that these results from *Yahoo* represent only the most populous features within the country.

4: *LOCODE*

LOCODE, also known as the "United Nations Code for Trade and Transport Locations" is managed by the United Nations Economic Commission for Europe but operates as a collaborative project across many international entities. *LOCODE* data can be obtained as a download in many formats or at `http://www.unece.org/cefact/locode/af.htm`. We chose to crawl this URL, obtaining 54 distinct features for Afghanistan.

Their primary attributes are described in Figure 144, with descriptions taken from `http://www.unece.org/cefact/locode/DocColumnDescription.htm`. An example of extracted values for three cities can be seen in Figure 145. Since *LOCODE*s contain specialized data concerning trade and transport locations, they provide relatively few populated place results within Afghanistan. The *Function* attribute, however, adds valuable information to populated places linked to a LOCODE *e.g.*, if the feature provides a rail terminal, an airport terminal or if the feature is a major postal exchange point.

Attribute	Kabul	Kandahar	Herat
woeid	1922738	1935376	1935285
placeTypeName	Town	Town	Town
name	Kabul	Kandahar	Herat
latitude	34.53091	31.61087	34.345081
longitude	69.136749	65.700279	62.186649
country	Afghanistan	Afghanistan	Afghanistan
admin1	Kabul	Kandahar	Herat
admin2	Kabul City	Kandahar City	Herat City
locality1	Kabul	Kandahr	Herat
locality2			
popRank	13	11	11
areaRank	4	3	2

Figure 143: A comparison of values gathered from the *Yahoo* dataset.

Attribute	Description
LOCODE (100%)	Alphabetic LOCODE identifier.
Name (100%)	Name of the place.
SubDiv (38.89%)	The ISO 1-3 character alphabetic and/or numeric code for the administrative division of the country.
Function (94.44%)	Contains a 8-digit function classifier code for the location.
Status (100%)	Indicates the status of the entry.
Coordinates (46.30%)	Geographic Coordinates in DMS format.

Figure 144: A table containing the property attributes collected from *Locode*.

5: *Wikipedia*

Wikipedia provides a wealth of structured information, not only from its infobox templates, but also from an in-context traversal of pages associated with a particular category. We used *Wikipedia*'s category structure to associate pages with the general "populated place" infobox template for specific countries by use of the 1^{st} and 2^{nd} order administrative division categories which contain sub-categories such as "Provinces of Afghanistan." We used this same structure to locate *Wikipedia* articles on populated places within Afghanistan by traversing the sub-categories contained within the *Wikipedia* category "Populated places in Afghanistan by province," located at `http://en.wikipedia.org/wiki/Category:Populated_places_in_Afghanistan_by_province`.

Once extracted, these pages could be parsed for useful information – namely the categories a given page belongs to and its infobox attributes. We were able to extract 844 such articles. Their primary attributes are described in Figure 146. An example of extracted values for three cities can be seen in Figure 147. We notice that even *Wikipedia*'s infobox is only semi-structured – that is, not all infoboxes contain exactly the same information, presented in exactly the same way, *e.g.*, Kabul (`http://en.wikipedia.org/wiki/Kabul`) has the population field titled "Population (2008) - Metro" while Herat (`http://en.wikipedia.org/wiki/Herat`) is listed as "Population (2006) - Total." These small textual differences make automated, mass extraction and aggregation of attributes slightly more challenging for *Wikipedia*. Fortunately, since the set of valid populated place pages has been aggregated, it is always possible to develop additional techniques to add more attributes from *Wikipedia*'s infobox templates as well as their category listings and unstructured text.

8.2.2 Gazetteer Linking

There are several gazetteer-linking projects currently active; a map of *NGA Unique Feature Identifiers* to *GeoNames geonamesids* has been compiled while *Yahoo* provides a public "concordance" API which maps between several datasets. Gazetteer linking is required to knowledgeably match features across datasets. This solves the problem of matching names where multiple features have the same name. These and other gazetteer linking methods will be described in the following sections.

Attribute	Kabul	Kandahar	Herat
LOCODE	AFKBL	AFKDH	AFHEA
Name	Kabul	Kandahar	Herat
SubDiv			
Function	- - 3 4 5 - - -	- - 3 4 - - - -	- - - 4 - - - -
Status	AI	AI	AI
Coordinates			

Figure 145: A comparison of values gathered from the *Locode* dataset.

Attribute	Description
WikiPageID (100%)	A numeric *Wikipedia* page identifier.
Article (100%)	Name of the article associated with the populated place.
Latitude (94.91%)	Latitude in decimal degrees.
Longitude (94.91%)	Longitude in decimal degrees.
Location (92.65%)	The country which the populated place resides in.
Population (06.87%)	Population figures.

Figure 146: A table containing the property attributes collected from *Wikipedia*.

1: *NGA* to *GeoNames*

As the *NGA* gazetteer provided the largest feature set, it was chosen as the "base" to which all of the other gazetteers will be linked. The *GeoNames* service provides a robust, tab-delimited mapping file between *NGA*'s *Unique Feature Identifiers* and *GeoNames*'s *geonamesids*. Once downloaded, we imported the mapping file into a SQL database for easy analysis. We then created an index table that mapped known *NGA* populated place features to known *GeoNames* populated place features within Afghanistan. We were able to make 32,111 such connections.

2: *Yahoo* Concordance Map

As part of its *GeoPlanet* service, *Yahoo* provides a "concordance map" API which, given a namespace and an ID, will return all other IDs associated with a given place. That is, if the concordance map API is sent *GeoNames* as the namespace and the *geonamesid* for Kabul, the *Yahoo* API will return all available IDs associated with the populated place Kabul. These include *LOCODE*s, *Yahoo*'s *WOEID* and *Wikipedia*'s *WikiPageID*, among others. We used *GeoNames* as our namespace in an attempt to match various IDs to the 32,111 records we had previously linked between the *NGA* and *GeoNames* datasets. We found, however, somewhat poor recall in almost all areas. Of the 32,111 *GeoNames* records, we were able to match 165 to the available 628 populated places discovered with *Yahoo*'s *WOEID*. We were able to match only 19 of the 54 total *LOCODE* entries and found that zero *Wikipedia WikiPageIDs* were linked.

3: *GeoNames* Alternate Names Map

On the *GeoNames* public forum, we located an extensive list of alternative name matchings between *GeoNames* and *NGA*, which also included *Wikipedia WikiPageIDs*. Using this file, we were able to link 79 *Wikipedia* pages to *NGA* and *GeoNames* features.

8.2.3 Gazetteer Aggregation

As was noted in the description of gazetteer attributes, several attributes overlap across multiple datasets. In cases where attributes overlap, we must choose which values will represent the "primary" value and which will be stored as "secondary" values. The final gazetteer should include as much information as possible, so that a null value from a dataset should never be assigned the "primary" value when another dataset successfully recalls a value.

Using these two guidelines, we settled on a "waterfall" method to assign primary values. For each attribute, we choose a hierarchy of datasets. Starting from the top of the hierarchy, we traverse downward until some dataset contains a non-null value for the attribute. We assign this first non-null value as the "primary" value and store the remainder as "secondary" values.

Attribute	Kabul	Kandahar	Herat
WikiPageID	16826	17260	14128
Article	Kabul	Kandahar	Herat
Latitude	34.53306	31.61694	34.34194
Longitude	69.16611	65.71694	62.20306
Location	Afghanistan	Afghanistan	Afghanistan
Population	2,850,000	468,200	349,000

Figure 147: A comparison of values gathered from the *Wikipedia* dataset.

Rather than define these hierarchies for each attribute, we chose to group attributes into three families and assign a waterfall hierarchy to each family. These families are defined in the following sections.

1: "Names" family

This attribute family contains items associated with a populated place's name and consists of the following attributes:

1. Primary Name
   - Primary name of the feature.
2. Secondary Names
   - Aggregate list of secondary names.
3. ASCII Name
   - Primary name in ASCII format.
4. Place Type Name
   - Type of place, *e.g.*, "Capital," or "Primary Administrative Division Seat."

Rather than collecting a single primary value for the secondary names, we aggregate secondary names from all datasets, then remove duplicate values from the aggregate list.

The values in the "Names" family are aggregated in the following order: the *NGA*, followed by *GeoNames*, followed by *Yahoo*. An example of the aggregate waterfall data for Kabul, Afghanistan is as follows:

- Primary Name
  - Kabul (Primary). Source: NGA
  - Kabul. Source: GeoNames
  - Kabul. Source: Yahoo
- Secondary Names
  - Cabool, Caboul, Cabul, Cabul - kabl, Cabúl, Caubul, Kabil, Kaboel, Kabol, Kaboul, Kabul, Kabula, Kabulas, Kabuli, Kabulo, Kabura, Kabúl, Kabûl, Kampoul, ... Source: Various
- ASCII Name
  - Kabul (Primary). Source: NGA
  - Kabul. Source: GeoNames
- Place Type Name
  - Town (Primary). Source: Yahoo
  - Capital. Source: NGA

2: "Location" family

This attribute family contains items associated with a populated place's geographic location and consists of the following attributes:

1. Latitude
    - Latitude portion of the feature's geo-coordinate.
2. Longitude
    - Longitude portion of the feature's geo-coordinate
3. MGRS
    - MGRS location of the feature.
4. ADM1
    - Level 1 Administrative Division where the feature resides.
5. ADM2
    - Level 2 Administrative Division where the feature resides.
6. ADM3
    - Level 3 Administrative Division where the feature resides.

The values in the "Location" family are aggregated in the following order: the *NGA*, followed by *GeoNames*, followed by *Yahoo*, followed by *Wikipedia*. An example of the aggregate waterfall data for Kabul, Afghanistan is as follows:

- Latitude
    - 34.516667 (Primary). Source: NGA
    - 34.52813. Source: GeoNames
    - 34.53091. Source: Yahoo
    - 34.53306. Source: Wikipedia
- Longitude
    - 69.183333 (Primary). Source: NGA
    - 69.17233. Source: GeoNames
    - 69.136749. Source: Yahoo
    - 69.16611. Source: Wikipedia
- MGRS
    - 42SWD1682719461 (Primary). Source: NGA
- ADM1
    - Kābul (Primary). Source: NGA
    - Kabul. Source: GeoNames
    - Kabul. Source: Yahoo
- ADM2
    - Kabul City (Primary). Source: Yahoo
- ADM3

- Unknown. Source: None

3: "Properties" family

This attribute family contains items associated with a populated place's property features and consists of the following attributes:

1. Population
   - The feature's population.
2. Elevation
   - The feature's elevation.
3. Function
   - The feature's function, *e.g.*, "airport" or "postal exchange."
4. Area
   - The feature's physical area.

The values in the "Properties" family are aggregated in the following order: *GeoNames*, followed by *Yahoo*, followed by *LOCODE*, followed by *Wikipedia*, followed by the *NGA*. An example of the aggregate waterfall data for Kabul, Afghanistan is as follows:

- Population
  - 3,043,532 (Primary). Source: GeoNames
  - 1,000,000-3,000,000. Source: Yahoo
  - 2,850,000. Source: Wikipedia
- Elevation
  - 1,790m (5,873ft) (Primary). Source: Wikipedia
- Function
  - Road, Airport, Postal Exchange (Primary). Source: Locode
- Area
  - 30-100 km^2 (Primary). Source: Yahoo

8.3 Sub-city features

8.3.1 Sub-City Feature Discovery

In addition to the physical attributes which we had previously aggregated, we identified another class of features that could be associated with a populated place: the sub-city feature. "Sub-city," in this instance, refers to any feature that resides "within" a given populated place, *e.g.*, roads, bus stops and religious centers. These features cannot always be linked explicitly to a populated place. For example, the "Hazrati Nabawi" mosque will not necessarily list its parent city as "Kabul." However, given a coordinate point sufficiently close to a populated place, we can intuitively establish a link via geo-location association.

Sub-city features provide important, high-resolution details about the interior of a populated place and can include geographic and non-geographic information, *e.g.*, roads, zoning districts and various buildings. Since high resolution analysis can only be as comprehensive as the dataset from which it is drawn, it is important that the coverage of sub-city features be as complete as possible. As a proof-of-concept, we assigned an analyst to survey several sources of sub-city features, specifically targeting Kabul, Afghanistan.

Figure 148: A comparison of mapped features from *Google* (left) to *Open Street Map* (right). Centered in Kabul, Afghanistan.

1: *Open Street Map*

One such collection of sub-city features belongs to the *Open Street Map* (*OSM*) group (`http://www.openstreetmap.org`). This open source project allows anyone to submit geo-tagged features in the form of "ways," "areas" or individual "points." *OSM* Ways are polylines that represent "lined" features: *e.g.*, roads, waterways and railways. *OSM* Areas are polygons that represent enclosed features such as mapped buildings, lakes and land-use information. *OSM* Points are single point features which can include bus stops, un-mapped buildings and mountain peaks. Features are added manually, through a user interface, or automatically, using a set of simple map-reading algorithms. See Figure 148 for a comparison between *OSM*'s mapped features and *Google*'s. We see that *OSM* has an overwhelming advantage when populating their map in this foreign locality.

1.1: Obtaining *OSM* Data

OSM data can be obtained via several channels, including its web interface and an API to *Nominatim*, the back-end on which the *OSM* mapping algorithms run. We chose to use a mass-extraction API provided by *Nominatim*, which allows a user to specify a large bounding box from which to collect *OSM* data.

For our purpose, we divided Afghanistan into three sections, 60-65 degrees east, 65-70 degrees east and 70-75 degrees east, each spanning 29-39 degrees north. We were able to use these smaller bounding boxes to extract *OSM* data in the XML format and store them in parse-able files.

OSM data is organized into three primary categories: "nodes," "ways" and "relations." Nodes are the primary building blocks of *OSM* data; each node contains a latitude/longitude point and a set of tags. These tags are usually descriptive information associated with a given point. For example the "Hazrati Nabawi" mosque is situated in Kabul and contains the tags, "religion: muslim," "amenity: place_of_worship" and "name: Hazrati Nabawi." Ways can either be polylines or polygons and consist of groups of nodes, referenced by an internal index. Relations can also either be polylines or polygons and consist of sets of ways and groups of nodes. Additionally, Ways and Relations may have separately associated tags. An example of this multi-level hierarchy can be seen in something as simple as a bus route: the entire route is recorded as a relation, with the stops as individual nodes and the paths between stops as ways. The route can be tagged with a name or number, the stops with location and paths with the names of traversed streets.

1.2: Organizing *OSM* Data

After simple XML parsing, we stored the *OSM* data in four SQL tables: "nodes," "ways," "relations" and "tags." We placed each *OSM* data type in its corresponding table and populated the tags table with all *OSM* tags. We also stored a reference index in the tags table in order to associate tags with their node, way or relation. *OSM* data is originally organized in a "top-down" method – that is, nodes are the primary building blocks and contain all of the geo-location information in the dataset. Since we are attempting to provide sub-city features "near" a given populated place, it would be difficult to determine whether a given way or relation should be displayed in the current data hierarchy, since we would have to check against the nodes and then determine whether any ways contain the selected node.

In order to optimize this search time, we created a "reverse-index" of the *OSM* data. That is, for each node, we generated a list of the ways and relations that contained the given node. We then stored these lists along with each node in the nodes table. Now, when selecting nodes "near" a given populated place, we have been provided with a list of ways and relations which should also be drawn on the map.

1.3: *OSM* Results

After traversing the *OSM* dataset, we obtained a total of 1,025,102 nodes, 58,150 ways and 347 relations. Of the 1,025,102 nodes, 4,683 do not belong to any way or relation, meaning that they are stand-alone features. These features are associated with 158,738 total tags.

Here are two examples of *OSM* features and tags within Kabul, Afghanistan are:

- Salang Watt Highway
  - oneway: yes
  - ref: A76
  - highway: primary
  - created_by: Potlatch 0.8a; JOSM
  - name: Salang Watt
- Rabia Balki Hospital
  - amenity: hospital
  - name: Rabia Balki Hospital
  - area: yes

2: *Wikipedia* features

We postulated that it should be possible to use similar "downward" traversal and extraction methods as in our prior *Wikipedia* gazetteer generation projects. Rather than focus on a single category, however, we located a few very broad categories for traversal: "Categories by Country" (`http://en.wikipedia.org/w/index.php?title=Category:Categories_by_country`), "Building and structure types by country" (`http://en.wikipedia.org/wiki/Category:Building_and_structure_types_by_country`) and "Buildings and structures in Afghanistan" (`http://en.wikipedia.org/wiki/Category:Buildings_and_structures_in_Afghanistan`). After traversing these categories, we stored all of their pages and sub-categories in a SQL database while removing duplicates. We then crawled all of those collected sub-categories to generate additional sub-categories and pages.

Once this process was complete, we had collected 46,518 combined pages and categories. We examined this dataset for any entry whose article or parent article contained the string "afghanistan" and gained a total of 508 categories. These categories usually corresponded to lists of geographic features within Afghanistan, *e.g.*, "Schools in Afghanistan" (`http://en.wikipedia.org/wiki/Category:Schools_in_Afghanistan`) and "Mosques in Afghanistan" (`http://en.wikipedia.org/wiki/Category:Mosques_in_Afghanistan`). We selected these categories for further traversal. From our prior "broad category" traversal and targeted Afghanistan traversal, we discovered a total of 1,898 pages associated with the country. Of these, 508 contained infobox templates with latitude/longitude coordinates. These 508 represent probable sub-city features within Afghanistan.

Two examples of sub-city features collected from *Wikipedia* are the "Salma Dam," (`http://en.wikipedia.org/wiki/Salma_Dam`) whose parent category is "Dams in Afghanistan," (`http://en.wikipedia.org/wiki/Category:Dams_in_Afghanistan`) which is located at the point (34.33083, 63.82528) and the "Afghanistan–Uzbekistan Friendship Bridge," (`http://en.wikipedia.org/wiki/Afghanistan%E2%80%93Uzbekistan_Friendship_Bridge`) whose parent category is "Bridges in Afghanistan," (`http://en.wikipedia.org/wiki/Category:Bridges_in_Afghanistan`) and which is located at point (37.2278, 67.4282).

3: *Afghanistan Information Management Services*

The *Afghanistan Information Management Services* (*AIMS*) is a project which is "building information management capacity in government and delivers information management services to organizations across Afghanistan." The *AIMS* project has created several shape files of features within Afghanistan, which are located at `http://www.aims.org.af/sroots.aspx?seckeys=69&seckeyo=44&seckeyz=37`. We extracted records from one such shape file, "Place of Interest - (Point)" within Kabul, Afghanistan as proof-of-concept.

After downloading the "Place of Interest - (Point)" shape file (`http://www.aims.org.af/services/mapping/shape_files/kabul/point/place_of_interest.zip`), we used a *Linux* utility called "shpdump" (shape dump), which takes an ESRI shape file as input and outputs a plain-text file of the features contained within the file. This file was then imported into an SQL database for use as sub-city features. We were able to capture 38 features such as the "Restauranti Haji Baba," a local restaurant, and "Munari Sayed Jamaludin Afghan," a historical site within Kabul. Since these features originated from a shape file, they were already encoded with their latitude/longitude locations.

Two examples of sub-city features collected from *AIMS* are the "Hotali Plaza," located at the point (34.51943, 69.17376) and the "Maqbara-i-Abdurahman," (34.52096, 69.1759).

4: *Liwal's Yellow Pages* and *ACBAR*

We located two similar business directory websites. The first was *Liwal's Yellow Pages* (`http://www.yellowpages.liwal.net/`), a website which purports to list businesses within Afghanistan. We extracted 478 features using our existing page-crawling technology. These entries contained semi-structured "tags," including items such as phone number and address. The second directory was from the *Agency Co-ordinating Body for Afghan Relief* (*ACBAR*), which contains a directory of non-government organizations (NGOs) operating within Afghanistan. We again used our existing page-crawling technology to extract 551 features. These entries also contain semi-structured tags, including a textual address.

Since both directory sites shared a similar structure, we chose to process them together. We used *Google Maps* geo-coding service to attempt to resolve the features' textual addresses into latitude/longitude points. Of these 1029 total features, we were able to successfully gather geo-coordinates for 954 of them. However, upon further examination, we discovered that of these 954 entries, we had only collected 72 distinct geo-coordinate points. Further, of these 72 points, only 6 actually exist within Afghanistan – the remainder are either incorrectly listed or have the address of their headquarters (in a different country) listed.

We believe that few distinct points were discovered because many of the collected addresses are imprecise, *c.g.*, "House No. 75, Street No. 9, Taimani Street, Kabul, Afghanistan," or "Jalalabad Road, Kabul/Afghanistan PO Box 54 Jalalabad Road Kabul, Afghanistan." These directions are helpful for a resident of the city, but do not provide enough resolution for *Google Maps* to correctly geo-code them. Instead, *Google Maps* resolves to a broader location; 867 records resolve to the broad coordinate for the city of Kabul, Afghanistan, itself.

Due to the inaccuracy of the geo-coding process, we chose not to use the records gathered from *Liwal's* and *ACBAR*. Further, unless a source provides well-structured addresses, it seems probable that the most helpful sub-city features must already contain their own geo-coordinate.

5: Additional Sources

Traditional source discovery for sub-city features can quickly become an exercise in long-tail theory; that is, the bulk of features will reside within a few very populous websites and, in order to collect the remaining features, many, many additional websites must be located. Rather than exhausting time spent on locating these sites which may contain only a few features each, the *Capstone* project turned to alternative sources.

In the mosque discovery task, we had created a *Media Aggregation Engine* which allowed multi-media website such as *Flickr*, *YouTube* and *Picassa* to be queried for media-items representing various features within a given area. We also developed a grouping algorithm which allowed for media items who shared a general feature-type to be grouped within a small area, allowing a higher confidence guess as to where a particular feature is. That is, if ten disparate media items report their coordinate as containing a mosque and can be grouped together, that point is more likely in fact a mosque rather than a single photo or video which might contain a false-positive or other error.

Using this same technology, we postulated that it should be possible to gather additional sub-city features for inclusion within the *Capstone* system. Since media-queries must be directed by keyword. we chose to investigate two feature types: mosques and schools. Further, since media-queries must also be given a geo-spatial bounding box, we chose to limit our initial investigation to the city of Kabul, whose coordinates are known from the aggregate populated places data.

In order to locate mosque points, we used the *Media Aggregation Engine*, developed in March, with the keywords: "mosque," "masjed," "masjid" and "islamic center." In order to ensure some accuracy, we also limited features to any group of two or more media items. In addition, we limited the media sources to only *Flickr* for this proof-of-concept test. Using these parameters, we were able to locate 11 total mosque features within Kabul, some of which can be seen in Figure 149. An additional point of interest within the figure is that some of the

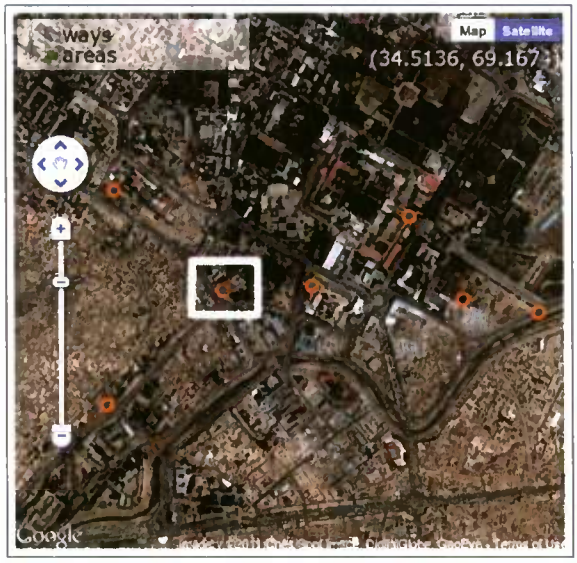


Figure 149: An overlay of *Flickr* Mosque features (orange) and *Open Street Map* Mosque features (brown).

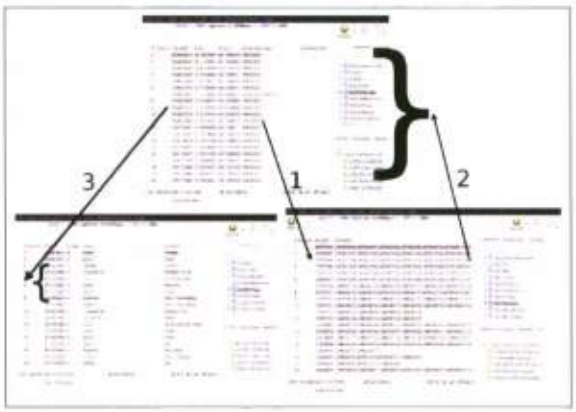


Figure 150: A diagram of the set of queries required to retrieve *Open Street Map* data from MySQL.

Flickr features fall very close to some of the *Open Street Map* points, as denoted by the white box drawn on the figure. This serves to reinforce the high-confidence with which features were selected by media items.

Using similar parameters with the keywords, "school," "university," "college," and "education," the *Media Aggregation Engine* was able to locate 5 additional school features within Kabul.

8.3.2 Sub-City Optimization

While examining the *Capstone* front-end interface, we noticed a major delay when drawing sub-city features on the *Google Map* viewport; in some cases it would take between five and ten seconds to draw a set of features. Upon further analysis, we noticed an inefficiency in the method used to collect this information which was related to our data-storage engine, *MySQL*. During our work in January, we noted that storing populated place data as a *MongoDB* document was much more efficient than as a set of *MySQL* tables. The sub-city features had not made this transition and were continuing to be stored within *MySQL*; as Figure 150 shows, the process of retrieving sub-city features from *MySQL* is non-trivial.

Originally, we stored the sub-city information in the following way: "nodes" are the building-blocks of each feature and are the only data type to contain latitude/longitude points, "ways" and "relations" are simply ordered

```
{
    "_id" : ObjectId("4d346ced1e56d0d606000000"),
    "id" : 22958150,
    "name" : "Amu Darya; Amyderya",
    "tags" : {
        "boat" : "no",
        "created_by" : "Potlatch 0.7a",
        "int_name" : "Amu Darya",
        "name" : "Amu Darya; Amyderya",
        "name:en" : "Amudaryo",
        "name:fa" : "آمودریا",
        "name:tg" : "Омударё",
        "name:tk" : "Amyderya",
        "waterway" : "river"
    },
    "nodes" : [
        {
            "id" : 247365906,
            "lat" : 37.6865001,
            "lon" : 65.3289706,
            "tags" : [ ]
        },
                ...
        {
            "id" : 476477994,
            "lat" : 38.3810486,
            "lon" : 64.3656079,
            "tags" : [ ]
        }
    ],
    "source" : "osm"
}
```


Figure 151: A sample sub-city document as retrieved from MongoDB.

sets of nodes which define a polyline or polygon. Further, each node and each way can be associated with a set of key-value tags which provide further information about the given feature. Thus, in *MySQL*, the process of selecting relevant sub-city features must begin with the "node" table, since no others provide a geo-spatial location. Once a set of nodes is selected by geo-spatial proximity to a given point, each node's list of parent ways or relations must be examined next, as seen in Figure 150 {1}. Once the appropriate way or ways are selected, the nodes table must again be examined to determine the entire set of nodes (fig. 150 {2}) associated with the particular way. Finally, the tags table must be checked for each node and each way in order to collect their key-value tags (fig. 150 {3}).

In order to collect all of the information for a way which contains 20 child nodes, the *Capstone* system must perform 43 separate queries: one to discover the initial set of nodes within a given area, one to locate the node's parent way, one per node within the way (20) in order to get a complete set of coordinates and finally one per node plus the parent way in the *tags* table (21) in order to collect all of the relevant key-value tags for the feature. This process becomes very cost-prohibitive, especially in areas with a high density of sub-city features. Further, as noted, each node within the given way must be queried individually since the set must always remain in the same ordered.

Rather than attempt to optimize this method of storage, we chose to move the sub-city feature data into *MongoDB*, similar to how the general attribute information associated with a populated-place is stored. In this way, all data associated with a given sub-city feature can be stored in one place, as seen in Figure 151. Each document within *MongoDB* contains all node and tag information for a given feature. In this way the *Capstone* can query the *MongoDB* collection just once for any way which contains a node which is within our specified geo-spatial proximity, that is, any feature containing a node within the bounds of the *Google Maps* viewport.

8.3.3 Categorization

The *Capstone* was presented with a *Google Maps* overlay which contained three selectable feature types, "ways," "areas" and "points." While it is certainly helpful to distinguish between these three feature types, we realized that it would be difficult to perform complex analysis on the set of features using only these attributes; for example, if

Figure 152: An overlay of categorized sub-city features within Kabul, displaying religious and restaurant features.

an analyst wished to view all of the mosques within Kabul, the analyst would have to decide if a mosque feature counted as a point or an area, perhaps choosing one or both, the analyst would then have to mouse over each presented feature, searching for a mosque.

In order to support complex analysis, the overlay must be able to distinguish features by key categories. After an initial survey of similar services such as *Google Earth* [27] and *WikiMapia* [20], we settled on the following list of categories:

- Ways:
  - "bridge," "highway," "natural," "railway," "waterway," "other"
- Areas:
  - "hospital," "bank," "cinema," "post office," "parking," "religious," "college," "university," "school," "restaurant," "natural," "amenity," "landuse," "building," "place," "power," "address," "public building," "populated place" and "other"

We also chose to drop the distinction between "areas" and "points" as there is no logical difference other than the method in which the feature is drawn on the map overlay.

Since each feature source lists information in its own way, the *Capstone* system needs a customizable algorithm for sorting features into each category. The *Capstone*'s current sub-city feature sources include: *Open Street Map*, *Wikipedia*, *Afghanistan Information Management Services* and *Flickr*. An example of the categorized sub-city map can be seen in Figure 152.

1: *Open Street Map* Categorization

As seen in Figure 151, *Open Street Map* features contain a list of key-value pair tags such as "amenity: restaurant" or "name: Kyber Restaurant." Using these tags, we built a set of filtering rules in order to classify each feature into its appropriate category. For example, the "religious" category examines these tags for keywords such as "mosque," "religious center," "islamic center" or "temple." If a tag matches a keyword, we classify the feature as the matching category. In this way, of the 10,473 total *Open Street Map* sub-city features, we were able to successfully classify 6,908.

2: *Wikipedia* Categorization

During the extraction of *Wikipedia* articles as sub-city features, the *Capstone* system captured article name and parent category name. These two attributes were used to group the *Wikipedia* articles into their specific categories,

similar to how the *Open Street Map* data was sorted. That is, if a given article belonged to the category "Mosques in Afghanistan," the *Capstone* system was able to identify the article as belonging in the "religious" category. Unfortunately many of the *Wikipedia* categories are very different than our own mapping categories, such as "Archaeological sites in Afghanistan" or "Prisons in Afghanistan," so only 13 of 220 captured *Wikipedia* features were categorized. This can be remedied by expanding the definition of some categories to include additional keywords or by expanding the complete *Capstone* categories list.

3: *Afghanistan Information Management Services* Categorization

The sub-city features extracted from the *Afghanistan Information Management Services* (*AIMS*) group were stored in shapefiles and pre-sorted into categories such as "restaurant" or "historic site." Since some of the *AIMS* categories matched our own, we were able to write a simple translation from their source to the *Capstone* system. In this way, all of the *AIMS* features could be theoretically categorized, but the current *Capstone* system does not use "historic site" as an explicit category; therefore, 11 of the 37 features collected from *AIMS* were placed into existing *Capstone* categories.

4: *Flickr* Categorization

Since feature-discovery on *Flickr* required the inclusion of keyword information, the *Capstone* system simply queried *Flickr* for two of its existing categories, "mosques" and "schools." This made categorization very simple as all discovered features were automatically placed within the correct category. All 16 *Flickr* features were placed into either the "school" or "religious" categories.

8.4 Real-time media

The *Capstone* system aggregates from many sources. As proof-of-concept, we chose three areas of study: adding news stories from *AlJazeerai*, *Topix.com* [39] and retrieving real-time news information from *Twitter* [19].

8.4.1 *AlJazeera* Articles

Stories are taken directly from the Middle Eastern news source, Al Jazeera. Stories are currently pulled from the page in real time using our page-crawling technology.

8.4.2 *Topix* Articles

Topix is a news aggregation service which draws from several sources. They present each story with its headline and summary as seen in Figure 153. Since each story shares the same structure regardless of source, it is relatively easy for the *Capstone* system, using *Cazoodle*'s web-crawling technology, to extract the article data in a structured format.

Similar to *Al Jazeera*, *Topix* provides an implicit populated-place sorting method. *Topix* sorts its articles by country and city, making it easy to extract relevant data for the *Capstone* system. A user must simply navigate to `http://www.topic.com/af` to view articles about the whole of Afghanistan. In order to retrieve higher-resolution data, we add the lowercase city name to the end of the URL, for example: `http://www.topix.com/af/kabul` will display articles related to Kabul and `http://www.topix.com/af/herat` will display articles related to Herat.

In this way, the *Capstone* system aggregates news from *Topix* in much the same way as from *Al Jazeera*. An example of this final aggregation can be seen in Figure 154. Note that stories are sorted by timestamp, regardless of source.

8.4.3 *Twitter* Real Time News

Twitter and other real-time news sources have been of increasing interest as an important source of live information. This has been seen recently in the conflict in Egypt with former president Hosni Mubarak when *Twitter* was used to spread information and organize protests [13, 14, 35].

Such real-time information is especially important to an analyst in order to understand the current ground situation in a given populated place. As such, integration with the *Capstone* project was necessary.

As proof-of-concept, *Twitter* was added to the *Capstone* interface using their public API. This API provides methods to query for users whose listed location is within a given radius of a latitude/longitude point. It also displays geo-tagged "tweets" from mobile devices such as an iPhone. Using this API, the *Capstone* is able to

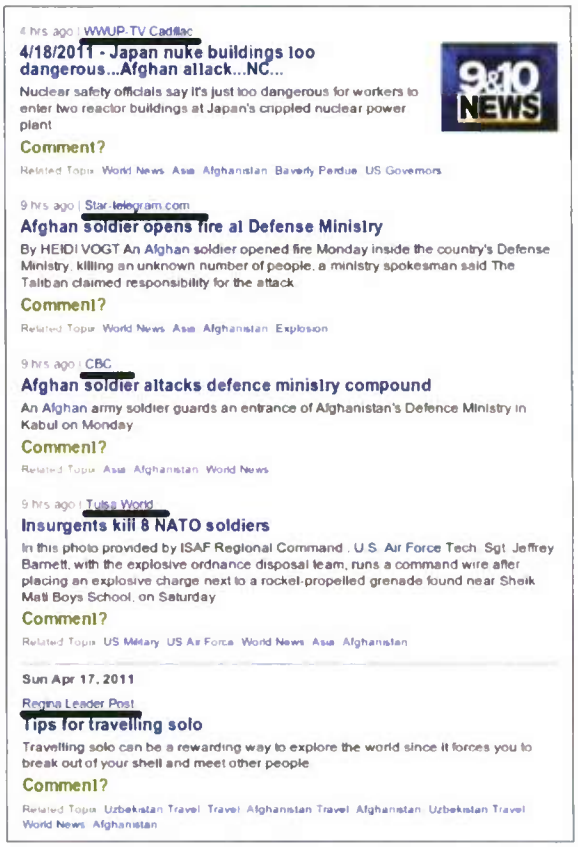


Figure 153: An example of *Topix* aggregate articles from Kabul with sources underlined in black.

list relevant real-time data for each given place. An example of such can be seen in figure 155. Note that these examples are primarily foreign language text. Further analysis might include translation to an analyst's native tongue and filtering based on input parameters, *e.g.*, "conflict" or "protest."

8.5 People collection

Collecting a list of people associated with a given populated place yields several analytical advantages. An analyst is able to use this list in order to gauge a populated place's relative importance; that is, if Hamid Karzai, the current President of Afghanistan, frequents a particular area, it can be said that, by association, that place must be important. Further, by knowing where particular groups of people congregate, an analyst can infer specific events in an area; that is, a government conference can be inferred by the presence of a group of high-ranking government officials visiting a particular place.

8.5.1 Populated Place Metadata – People

1: Name collection

In order to create this geospatially-linked person directory, we began parsing data from *AlJazeera* articles. Articles from *AlJazeera*, as seen in Figure 156, contain a footer which lists all of the entities mentioned therein, including people, countries, cities and organizations. By focusing on the city of Kabul, we were able to obtain 697 distinct news articles, which contained 587 individual names (via these entity lists).

From this collection of people, we created a *MongoDB* [31] collection, with a document for each distinct name. *MongoDB* is a schema-less, "no-SQL" database, which allows aggregate data to be stored within unified "documents." These documents are flexible storage mechanisms that can be modified at run-time without the overhead requirement of modifying an underlying schema as might be required by a traditional relational database.

Figure 154: An example of aggregate news stories for Kabul.

Such a flexible storage mechanism allows us to store, in a free-form manner the text and entity lists of each article that is associated with a given person, as well as the list of people associated with a given place. As shown in Figure 157, we can use the same data-store to represent data from multiple perspectives – that of each person we have discovered and that of each source article.

2: Name results

Of the list of 587 names as collected from articles linked with the city of Kabul, only 18 appear more than 10 times. This subset can be seen in Figure 158: as expected, familiar names such as "Hamid Karzai" and "Barack Obama" are mentioned the most. The names are split in origin between the United States and the Middle East and contain seven political leaders, six journalists, four military leaders and one diplomat.

An additional note of interest is that the name "Hoda Abdel-Hamid" also appears as "Hoda Abdel Hamid." Since our collection method was naive about some spelling differences – assuming that each name entry was entered correctly and that a single name convention was followed – we have picked up several duplicates. Additional analysis would be required to differentiate names that are meant to be separate entries and which should be grouped together.

Similarly, well-known places are also mentioned the most. Cities, however, are much more dramatically clumped together: only six have more than 10 mentions. Since city names are, perhaps, more standardized than people's names, we do not find the same multi-entry problem as with the prior set of names. Two of these cities, Washington and New York, come from the United States, while the rest are various foreign locales.

Additionally, these names might be further linked by way of shared articles, cities or organizations. That is, if a name is mentioned alongside a city, it can loosely be assumed that the name is associated with that city. Further, if two names appear alongside the same city, it can also be loosely assumed that there exists some link between those two people. Finally, if two cities are often mentioned together, a link might be assumed between these locations. That is, if Washington, DC and Kabul are often mentioned together, an analyst might assume that there is a reasonable amount of interaction between the people of these cities.

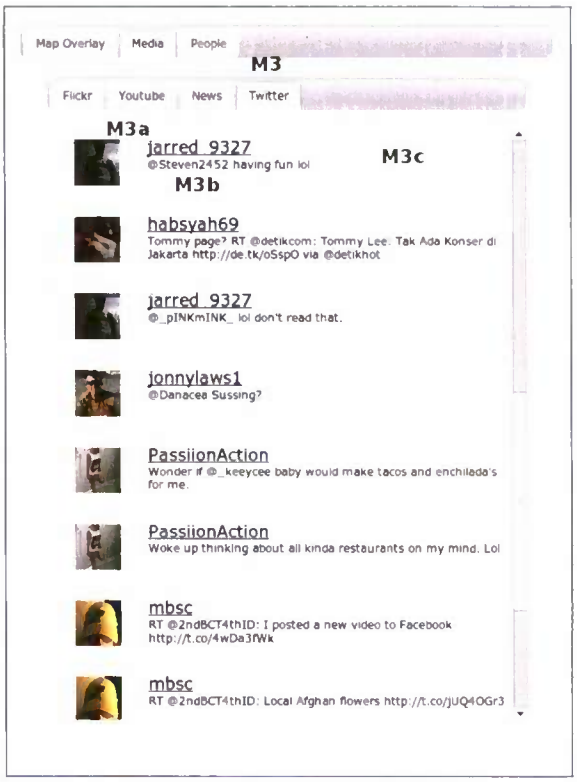


Figure 155: An example of *Twitter* data originating from Kabul.

8.5.2 Additional Sources

The *Capstone* system has used *AlJazeera* news articles to link people to populated places by identifying a person whenever they are mentioned alongside a place. These general methods can be applied to linking one person to another: that is, creating a link whenever two people are mentioned in the same article. Several additional sources, including *MSNBC* and *http://www.defense.gov*, have also been used to reduce single-source bias and increase the overall "connectedness" of our person-person linking graph. The details of linking and source extraction can be found in Section 8.5.3.

We have used these sources to create a well-connected "person-graph," which maps the connections between different people that are associated with populated places in Afghanistan. This graph allows us to calculate shortest-path traversals from one person to another: this is very similar to the ideas of "six degrees of separation" in popular culture and which was originally explored by the paper, "An Experimental Study of the Small World Problem" [96] or "collaboration distance" in the academic sphere [21]. Additionally, we have used this graph to

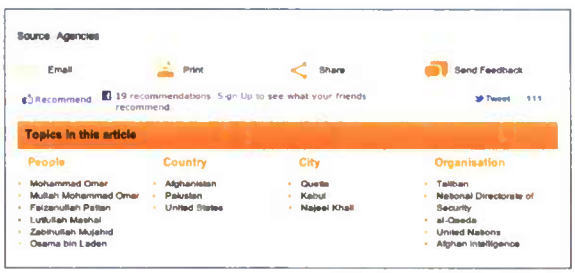


Figure 156: A footer to an *AlJazeera* article, showing various lists of entities contained therein.

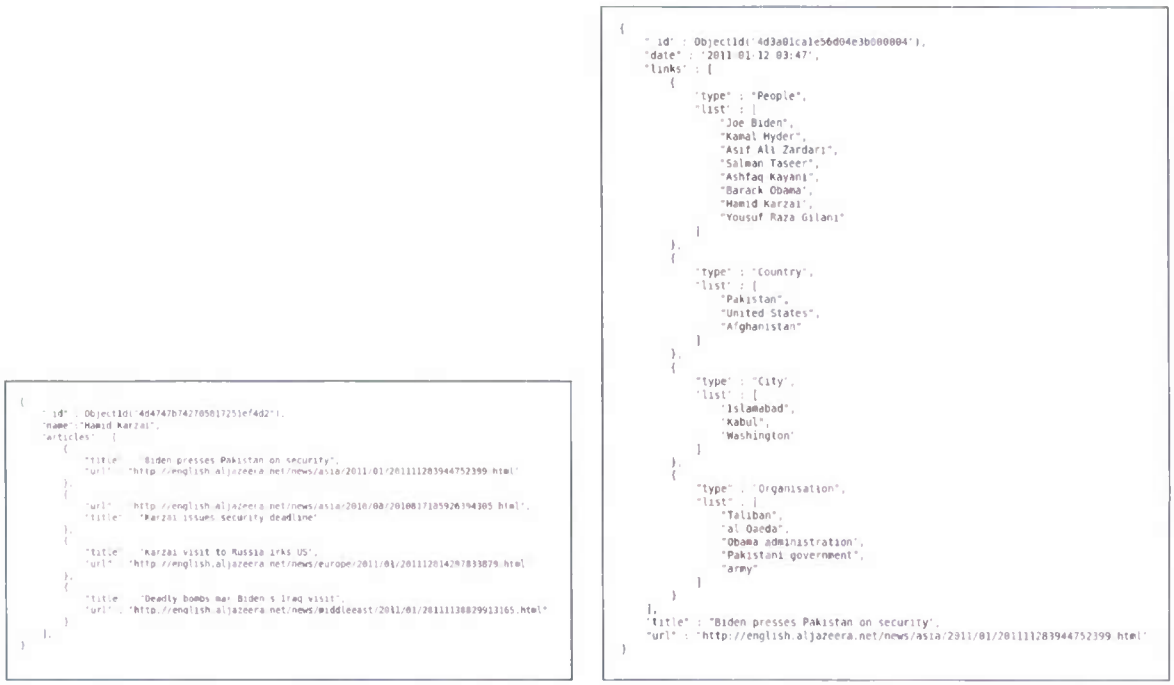


Figure 157: An example of two different *MongoDB* documents. One (left) is from the perspective of a single person and lists all associated articles. The second (right) is from the perspective of a single article and lists all associated entities.

Hamid Karzai	193
Barack Obama	56
James Bays	55
Sue Turton	33
Stanley McChrystal	28
David Petraeus	27
Zeina Khodr	19
Hillary Clinton	16
Robert Gates	16
Richard Holbrooke	16
Abdullah Abdullah	15
Gulbuddin Hekmatyar	14
Karl Eikenberry	13
Hoda Abdel-Hamid	13
David Chater	13
Hoda Abdel Hamid	11
Mohammad Omar	11
Zemarai Bashary	11

Kabul	633
Washington	49
Kandahar	46
Islamabad	28
London	24
New York	11

Figure 158: The list of names (left) and cities (right) that appear more than 10 times throughout the *AlJazeera* articles that were collected.

build "micro-consoles" which allow an analyst to explore different aspects of the linked people-data. Section 8.6.6 describes some of these consoles, which include: source-article lists of links generated by each source, a list of linked cities, a "Co-Mentions" graph which plots the first-order connections of a given person, a "path" console which allows an analyst to view the linked steps between two given people and a timeline which allows an analyst to view a given person's temporal importance.

8.5.3 People-Name Extraction and Linking

1: Name Collection

We have detailed how people-names could be extracted and linked to populated places within Afghanistan through the use of *AlJazeera*'s news articles. In short, articles furnished by *AlJazeera* contain several lists of entities which have appeared within the article, namely: people, countries, cities and organizations. Figure 156 shows one such list; in this article which describes the deaths of aid workers in Afghanistan, we can see that the United States and Afghanistan are mentioned, as well as some of each country's high-ranking officials. Using these lists, we were able to extract 587 individual person-names, which were linked to the city of Kabul.

2: Linking with *AlJazeera*

By inspection, we found that when two people are mentioned within the text of a given article it is reasonable to assume a link between them. For example, in a recent story on election fraud in Afghanistan, the names *Hamid Karzai* and *Fazel Ahmad Manawi* are used [11]. Using our linking algorithm, we create and store a link between these two people.

Upon further inspection, we find that Hamid Karzai is the current leader of Afghanistan and that Fazel Ahmad Manawi is the chairman of Afghanistan's Independent Election Commission. This is an example of a strong link, as both of these people are actually linked through their positions in the Afghanistan government. Some links are weaker: in another recent article, *AlJazeera* interviewed members of the Afghan-American community for their reflections on how the events of September 11^{th}, 2001, have shaped the political landscape [10] in the United States. In this article, the many names of the interviewed are listed. Upon further inspection, we find that, apart from the fact that these people belong to the Afghan-American community, there is no direct link between them.

Therefore, we title these links as "naive" because they do not imply anything regarding the strength, directionality or context of the association between two people. Upon initial analysis, however, we find that even weak links such as those between the members of the Afghan-American community can be used to describe useful connections between groups of people.

Using these methods, we were able to gather 5,899 links among 548 names.

3: Linking with *MSNBC*

Through the inclusion of multiple sources, we are able to increase our confidence in assigning a "link" between two people. Further, by including sources from different localities, we are able to obtain an additional perspective on the interconnections between the people we are working with. For example, *AlJazeera*, which is based primarily in the Middle East, is a great source for people whose influence is strong in that area. By adding links obtained through *MSNBC*, we can also capture those whose influence is strong in the Western world.

The structure of an *MSNBC* article is slightly different than that of an *AlJazeera* article: *MSNBC* does not provide an explicit list of those entities which are mentioned in their articles. This makes collecting novel names from *MSNBC* very difficult, as there is no baseline from which to compare or discover new entities. Further, the text of each article is less relevant for our linking purposes, as it would be quite expensive to collect the content of each article associated with each person before attempting analysis.

In order to create links with *MSNBC* articles, we discovered a simpler method. For each person in our *AlJazeera* dictionary of names, we perform a search on *MSNBC*'s website, with its name as the query. We then store all resulting article titles from the first ten pages of results, along with their name, in our database. We restrict the results to the first ten pages to ensure a reasonable level of relevance to the initial search query – and also because it would be prohibitively expensive to exhaustively collect and analyze all articles: for instance, "Barack Obama" has approximately 573,000 article titles associated with him alone.

Once these titles are collected, we create a link between any two people whose name-based queries resulted in the same article title appearing in their results. In this way, we assume that articles that appear in a name-based query must somehow be associated with that person. That is, a search on *MSNBC* for "Barack Obama" should only return results that are somehow relevant to that person. Additionally, as in the *AlJazeera*, we assume that if two people are mentioned in the same article (or share an article title) that there is some naive link that can be used to associate them.

As in *AlJazeera*, we see that these assumptions hold true: in an article regarding al Qaeda and the United States, we find both "Hillary Clinton" and "Barack Obama" mentioned in the article's text, which causes the title "Clinton: al Qaeda behind unconfirmed threat to U.S.," to appear in queries for both people [33]. Using these methods, we were able to gather 26 links among 20 names.

We postulate that these numbers are lower than those gathered from *AlJazeera* for two primary reasons: 1) Our collection methods with *MSNBC* are far less exact than those with *AlJazeera*, since the latter is able to provide explicit lists of the entities which are mentioned in their articles while the *MSNBC* matching is much more "fuzzy," and 2) the collection of article titles from *MSNBC* was undirected; since the initial queries on *MSNBC*'s website did not consider existing or potential matches, the title collection was naive, which greatly reduced the chance that two titles might match.

4: Linking with *Defense.gov*

Although *AlJazeera* and *MSNBC* are based in different localities – and can provide different perspectives on the links between various people – they are both explicitly "news" sources and thus share similar problems in the ambiguity of their links. In order to counteract this effect, we included one additional source, *Defense.gov*.

One portion of *Defense.gov* is dedicated to photographs that have been taken by various military or non-military photographers. Accompanying each photograph is a short, single-paragraph description of its contents. We found that many of these photographs contain the names of people that we have already collected through *AlJazeera*. For example, one photo, which can be seen at the following URL: `http://www.defense.gov//Photos/NewsPhoto.aspx?NewsPhotoID=14728`, shows several members of the United States Military along with what appears to be two government officials. The descriptive text reads:

> Secretary of Defense Leon E. Panetta escorts Danish Defense Minister Gitte Lillelund Bech through an honor cordon and into the Pentagon on Aug. 17, 2011. Panetta and Bech will hold talks on a broad range of security issues. DoD photo by R. D. Ward. (Released)

By inspecting this description we see that two people are mentioned: Leon E. Panetta and Gitte Lillelund Bech. Since these two people appear in the same photograph, it is reasonable to assume that they have met or, at the very least, attended some event together. In this photo we see that it is an explicit link: "Panetta and Bech will hold talks on a broad range of security issues." Since these types of photographic links provide a much narrower context, they provide a significantly stronger connection between their two entities.

Since the *Defense.gov* website does not provide any sort of "search" functionality on their photos – and since they do not explicitly list the names of people included in each photograph, entity extraction becomes slightly more difficult.

To start with, we downloaded and stored a reference to each image and that image's description. Next, we iterated through our dictionary of names, as collected through *AlJazeera*. For each name, we examined the descriptive text of the image to see if that name appeared within. Because there are often subtle spelling differences in two instances of a single person's name, we used a slightly "looser" concept of matching – very similar to our previous work on semantic linking. We used a ratio of Levenshtein distance to word length in order to find matches. Levenshtein distance is a measure of the "edit distance" between two words [30], that is, the number of insertions and deletions required to transform one word into another.

Since names are generally composed of multiple words, we broke each photograph's description into tokens based on the " " (space) character. We also broke the target name into tokens based on spaces. We then iteratively examined each token in the description against the first token of the target name. If we found a match, we examined the following token against the next token in the target name, continuing until they no longer matched (not a valid name match) or we ran out of tokens in the target name (all tokens match, so therefore we have found a valid match). We defined a "match" as that of a Levenshtein distance to average word-length ratio of less than or equal to 0.2. That is, if an average of 1 in 5 or fewer characters needed to be changed in order to transform the first token into the second, then we consider those tokens as matching.

Through these *Defense.gov* images, we were able to create 620 links among 38 people. As expected, these numbers are considerably smaller than those found in *AlJazeera*

8.5.4 Linking conclusions

In all, we were able to collect 6,545 links among 550 distinct names. The most populous link was between the names "Hamid Karzai" and "Karzai." Inspection shows that this actually reflects two forms of Hamid Karzai's name, which points out an inherent problem in our name collection: by using *AlJazeera* as our sole source of names, we are greatly effected by inaccuracies in their articles and the lists that are generated by these articles. One additional erroneous entry was that of "Abraham Lincoln," whose name was incorrectly categorized as a

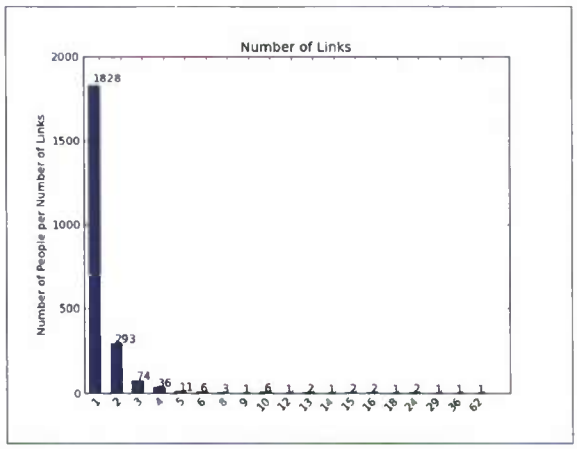


Figure 159: A binning of the number of links between people.

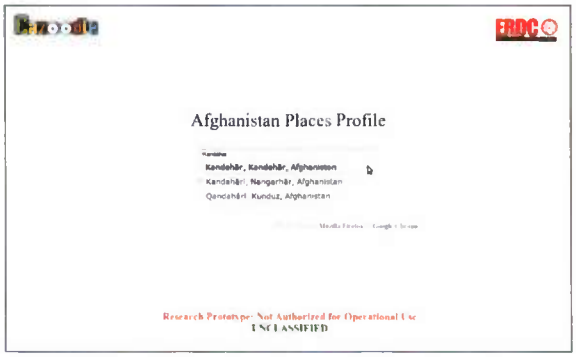


Figure 160: The Capstone Front-End's home page, highlighting the populated place name auto-completion for "Kandahar."

person when, in the article, the author was referring to the USS Abraham Lincoln, an aircraft carrier in the United States Navy.

The second most populous link was that between Hamid Karzai and Barack Obama, who shared 29 unique links. Figure 159 shows the number of links between people in our dataset. We can see that a large number of people share only one linking media, while some share two pieces.

8.6 Operation console

The Afghanistan Places Profile Capstone system is a one-stop shop for information about places in Afghanistan, collecting information from a variety of sources and visualizing the information in a dashboard interface. The system integrates traditional geospatial data, such as maps, written descriptions, and facts about places, with real-time information about weather, news, photos, videos, and people. The system integrates the technologies and lessons learned in Cazoodle's Phase II Small Business Innovation Research project.

8.6.1 Query Interface

The Capstone Front-End begins with a search-oriented home page, as seen in Figure 160. This portal allows a user to search for a populated place within our dataset by beginning to type the name of that place. Our system then performs a series of on-the-fly queries which supply a set of "guesses" of the full place-name to the user. The user then chooses one such place and proceeds to view the populated place's various aggregated attributes, displayed within a "detail" page. These "guesses" are powered by a JavaScript action called Asynchronous JavaScript and

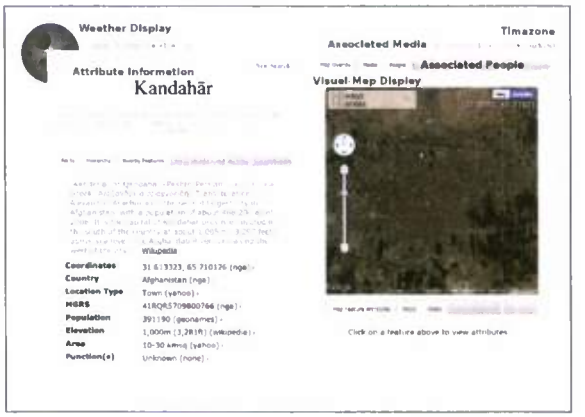


Figure 161: An example "detail" page for the city of Kandahar.

XML (AJAX). These on-the-fly queries use AJAX technology to perform partial searches on the primary and secondary name attributes of populated place features within the *Capstone* system.

Broadly, the detail page, as seen in Figure 161, is divided into six logical sections. On the left, we feature various attributes directly associated with the given populated place (Attribute Information). Above lies a snapshot of the current weather conditions around the populated place (Weather Display). On the right, we feature sub-city (Visual-Map Display) features, media attributes (Associated Media) and people (Associated People) associated with the given populated place by geographic proximity. Finally, we display relevant timezone information (Timezone) in the top right area.

We have chosen the city of Kandahar, located in the Kandahar province of Afghanistan, as our example city. These same instructions will apply generally to any populated place in our dataset.

The search portal can be accessed at `http://geoengine.cazoodle.com/capstone`. To follow the examples, search for "Kandahar" and select the first result.

8.6.2 Attribute Information

The associated attributes section has four distinct areas, highlighted in Figure 162. The sections marked A3 and A4 are taken from within the same "tabbed" area as seen initially in the section marked A2. The purpose of each area is described in the following subsections.

A1: Names

This section contains the primary and secondary name attributes for the selected area. Its individual parts are referenced in Figure 162, section A1 and are as follows:

A1a The feature's primary name.

A1b A click-able button that allows the user to specify a new feature name to query, similar to the search-oriented home page.

A1c A list of secondary names, aggregated from all associated feature gazetteers.

A2: Facts

This section contains various attributes collected from our aggregate dataset. We display the primary-chosen value for each attribute, along with its original source. We also allow the user to view the secondary values for each attribute, where they exist. Its individual parts are referenced in Figure 162, section A2 and are as follows:

A2a The first paragraph of the *Wikipedia* entry associated with the populated place, if it exists. Also links to *Wikipedia*.

A2b The title and type of an attribute.

Figure 162: A map of the "left" side of a populated place's detail page, including all sub-sections of the tabbed area.

A2c The primary and secondary values of an attribute. Secondary values can be viewed by clicking the small gray triangle to the right of the attribute value, which then expand as shown.

A3: Hierarchy

This section contains primary ADM level 1-3 information associated with the populated place.

A4: Nearby Features

This section contains a list of nearby populated places. It currently shows the name of each place in order of ascending distance, along with that distance and the cardinal heading to each populated place.

8.6.3 Weather Display

This section contains a snapshot of the current ground weather conditions for the populated place, if available. Weather conditions are cached within a radius of 0.5 decimal degrees. Cached records expire after an hour. Weather information is queried from *World Weather Online* (`http://www.worldweatheronline.com/`). This section's individual parts are referenced in Figure 162, section W1 and are as follows:

W1a A graphical representation of current weather conditions. Pictured is "clear night sky."

W1b A textual representation of the weather conditions. Displays the current temperature, daily high and daily low in degrees Celsius.

Figure 163: A map of the "right" side of a populated place's detail page, including all sub-sections of the visual-map tabbed area.

8.6.4 Visual-Map Display

The Visual section contains features that can be mapped on top of an embedded *Google Maps* map. Currently this consists of feature polylines, polygons and points. Polylines are referenced as "ways" in the map, while both polygons and points are listed as "areas." Its various functions and sub-sections are described as follows:

V1: *Google Map* controls and Mapped Features

This section contains the map and features high-level sub-city drawing controls as well as standard *Google Maps* zooming and map-type controls. Its individual parts are referenced in Figure 163, section V1, and are as follows:

V1a Standard *Google Maps* map-type controls. Allows the user to switch the base-map between a standard "map," with known roads and a "satellite" view.

V1b Standard *Google Maps* zooming and panning control. Users can also zoom the map with the mouse's *scroll wheel* and pan the map by clicking anywhere within the map and dragging, as if "pulling" the map.

V1c Represents the current location of the mouse-cursor over the map as its geo-coordinates in decimal degrees.

V1d High level controls allow a user to specify, broadly, what type of sub-city features should be drawn on the map.

V1e Features within the map are click-able entities. Once clicked, details concerning the feature will appear in area V2. The feature displayed at V1e is a "way," more specifically the "Kabul - Kandahar Road."

V2: Map Feature Information

This section contains information about the map features displayed in the *Google Maps* overlay above. Once a mapped feature is clicked, all relevant tags and other information are displayed here. This section is referenced in Figure 163, section V2.

V3: Additional Map Information

This is the same section as described in V1. It is referenced in Figure 163, section V3. Additional features are described as follows:

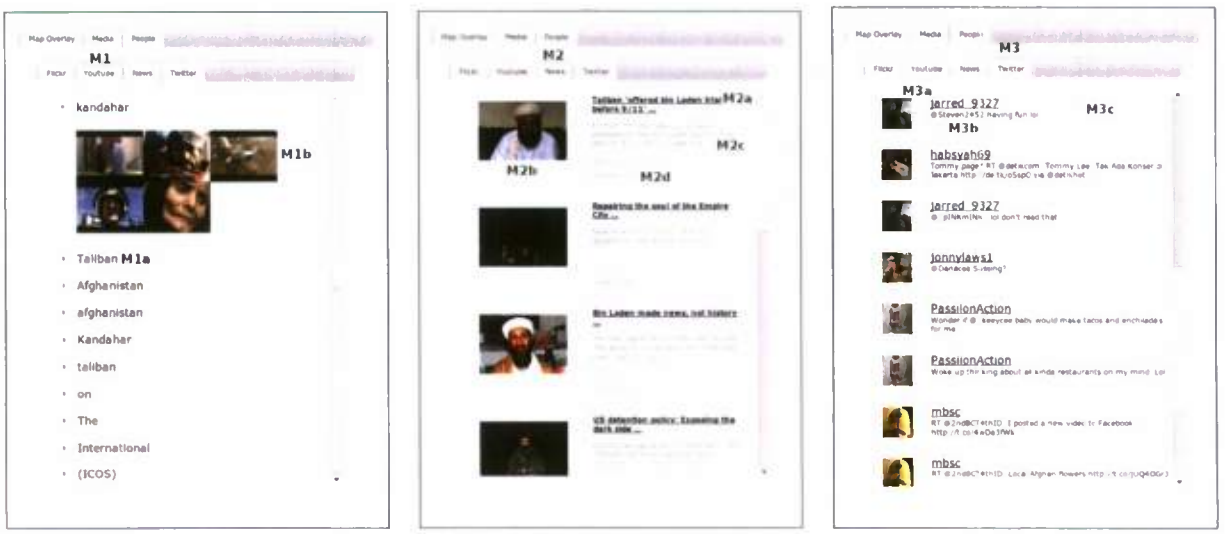


Figure 164: The sub-sections contained within the "Media" tab from the right-hand side of the *Capstone* front-end.

V3a An example of additional sub-city features selected to be drawn on the *Google Maps* overlay. Features can be selected and deselected by simply checking the box next to each item.

V3b This light green polyline is an example of a "Way."

V3c This blue circle is an example of a "Area."

V4: Map Features Tree

This section contains named features drawn within the map represented in a textual "tree" structure, with the name of each feature as a "root" and its attributes as its leaves. Its individual parts are referenced in Figure 163, section V4, and are as follows:

V4a An example of a named feature that is drawn within the above overlay. Clicking on the small triangle next to the name will expand its attributes, while clicking on the name itself will cause a marker to appear on the overlay, as seen in V4c.

V4b An example of the attributes associated with a sub-city feature currently drawn on the map. These generally take the form of "key-value" tags.

V4c A marker drawn on the map representing a feature named in section V4a.

8.6.5 Associated Media

This section contains the collected media associated with a given area. It aggregates from *Flickr* and *YouTube*, the *AlJazeera* news service and *Twitter*.

M1: *YouTube*

This section contains media items taken from *YouTube* and *Flickr*. Media items are discovered using our *Media Aggregation Engine*, which queries media items which are geo-tagged near the current location. Both the *YouTube* and *Flickr* sections share a similar structure. This structure is described in Figure 164 *M1* as follows:

M1a Media items are sorted by tag. That is, all media items within the "Afghanistan" section feature "Afghanistan" as one of their media tags.

M1b Media items are listed within their tag sections. Clicking on a media item will open a new window with the item's original web-page.

M2: *AlJazeera* News

This section contains news stories associated with the selected populated place. Currently stories are taken directly from the Middle Eastern news source, Al Jazeera. Stories are currently pulled from the page in real time using our page-crawling technology. The individual parts of the news section are referenced in Figure 164 *M2* as follows:

M2a The headline of each news story. Clicking on this will open a new window with the article's original web-page.

M2b A picture, if available, associated with the news story.

M2c A brief description of the news story.

M2d The date and time and source for each news article. Articles are sorted by date and time.

M3: *Twitter*

This section contains *Twitter* data which is pulled from the web in real time. *Twitter* "*Tweets*" appear here via one of two methods: 1) the "Tweet" itself has been geotagged near the current location or 2) the *Twitter* user reports their location as being near the current location. The individual parts of the *Twitter* section are referenced in Figure 164 *M3* as follows:

M3a The *Twitter* user's profile picture and user name. Clicking on either will open a new page with the details of the "Tweet."

M3b The text of the user's "Tweet."

M3c The date and time when the user's content was posted.

8.6.6 Associated People

The Associated People section contains a list of all of the people who have been linked to the given populated place through semantic linking via news articles and other media sources. Figure 165 shows this list, sorted alphabetically. Clicking on a name will open a new window with seven "miero consoles" which give different views on the aggregated linking data.

The first three simply display and link to the media items which contain their name from *AlJazeera*, *MSNBC* and *Defense.gov*. The fourth displays a list of all of the cities which appear in the same articles as the person, along with the frequency of their mentions. The fifth view, "Co-Mentions" shows a graph of the top 30 most populous links from the active person and their interconnections. The sixth is an interactive console for plotting the "path" from one name to another. The seventh console shows a timeline-view of the person's popularity from each of our three sources.

Consoles 1-3: Media View

As seen in Figure 166, the media-viewing consoles display exactly which media items are associated with each person. In this example, we see a list of articles and photographs which are related to Hamid Karzai. Clicking on any of these media items will open a new window with the item's original source.

Console 4: City View

The city view displays the frequency with which a person is mentioned alongside a particular city. As Figure 167 shows, Hamid Karzai is very well known in Kabul – he has far more mentions there than any other single city.

Console 5: Co-Mentions

The Co-Mentions console displays a visualization of how the people who were collected via media items are interconnected. As seen in Figure 168, this console displays top 30 links for a given person. Additionally, the console displays all of the "interior" links which connect those people who appear in the Co-Mentions graph. Further, the edges are visually weighted so that thicker lines correspond to links from more media items. The Co-Mentions graph can be manipulated by dragging the colored nodes so that their labels can be seen more clearly.

On the right of the Co-Mentions console, the list of people who appear in the graph are displayed. Clicking on one of these names will open a new set of miero-consoles with that person as the central figure.

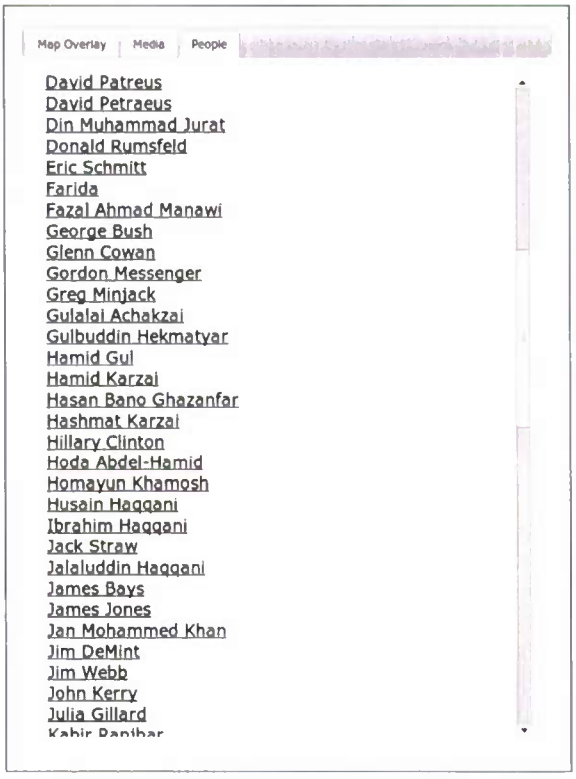


Figure 165: The list of people who are associated with the current populated place.

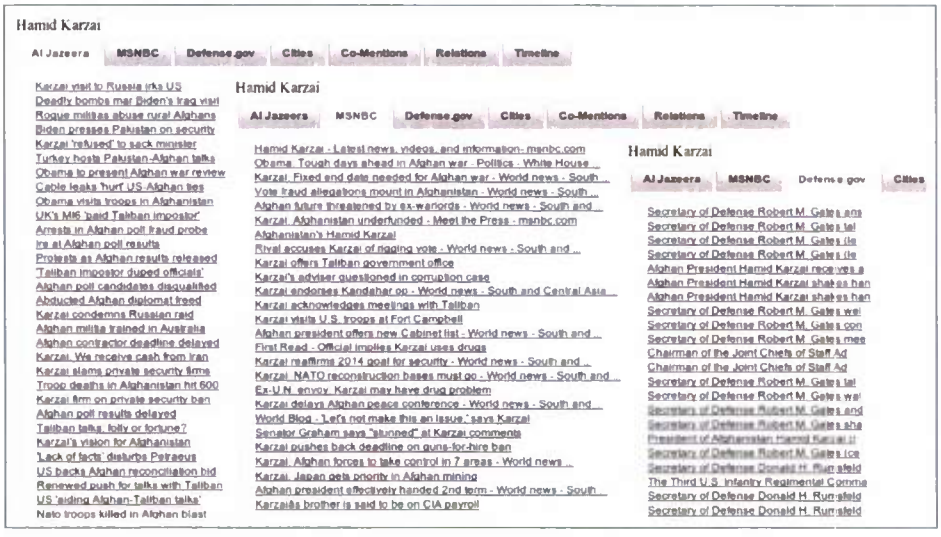


Figure 166: The *AlJazeera*, *MSNBC* and *Defense.gov* viewing consoles.

Console 6: Path Discovery

The Path Discovery console allows the path between any two people who have been collected via media articles. Upon typing a name query, the system will use an on-the-fly auto-complete algorithm similar to that of the search-oriented home page to automatically generate a list of appropriate names which match the user's query. After selecting a name, a list of the linking steps between the queried name and the currently active person will be

Kabul	633
Washington	49
Kandahar	46
Islamabad	28
London	24
New York	11

Figure 167: The cities with which Hamid Karzai is associated.

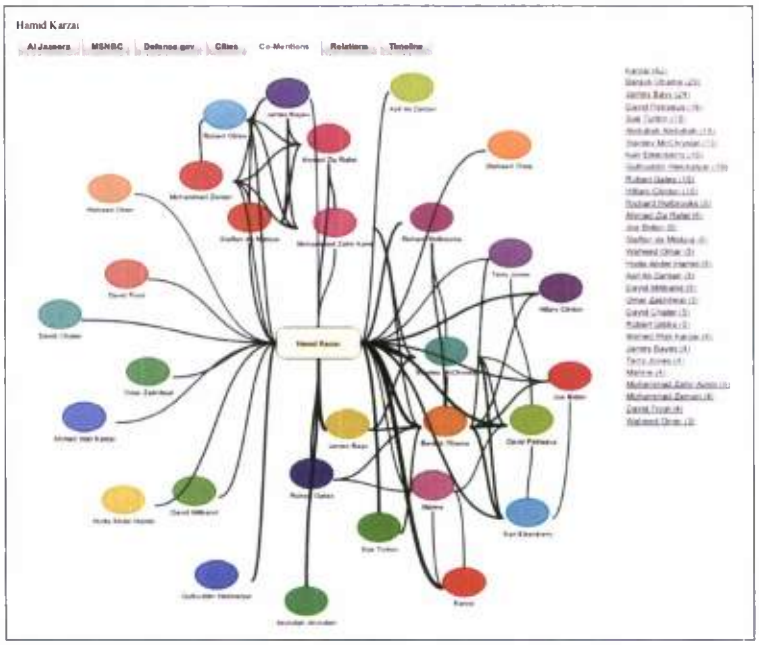


Figure 168: The Co-Mentions graph for Hamid Karzai.

displayed. As Figure 169 shows, these steps include a list of the linking evidence with which the two people are associated.

Console 7: Timeline

The timeline console shows the temporal relevance of the active person. As Figure 170 shows, the media items which are related to the active person are placed on the timeline based on their creation date. These media items are differentiated based on their source: the green flag represents media from *AlJazeera*, the gray square represents media from *Defense.gov* and the green flag represents media from *MSNBC*. The individual parts of the timeline are referenced in Figure 170 as follows:

T1 By clicking and dragging horizontally an an "empty" area in the timeline, the operator can scroll through the years that the timeline covers.

T2 The bottom-left section shows the name of the active person. Additionally, the operator can click on "Legend" to see a description of each type of media that is plotted (above). Clicking on these media items will filter the timeline by source.

T3 An individual media item. Clicking on it will open a window with more information: see *T4*

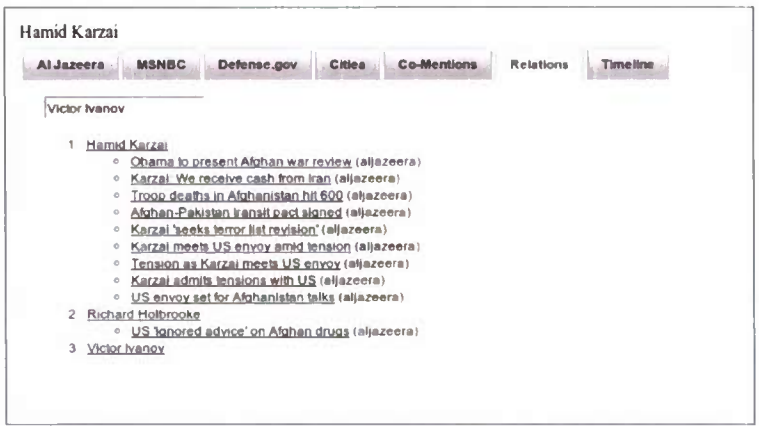


Figure 169: The path which links Hamid Karzai to Victor Ivanov.

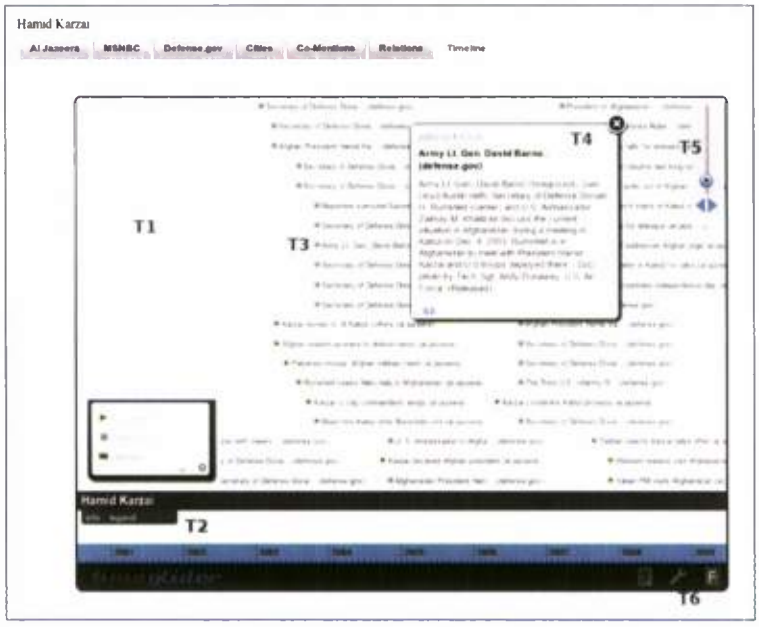


Figure 170: A timeline of articles relating to Hamid Karzai.

T4 The detail view of a single media item. The title, description and a link to the original source are displayed.

T5 The zoom slider allows the operator to change the scale of time which is shown in the timeline.

T6 Additional filtering controls. Clicking on the large gray "F" will allow the operator to filter by article title and source.

8.6.7 Timezone

As seen in Figure 171, the timezone section features three types of timezone information. First, the current *Universal Coordinated Time* (UTC) is displayed as a 24 hour clock. Second, the current local time is displayed as a 24 hour clock. Finally, a textual representation of the current timezone is displayed; in this case, "Asia/Kabul."

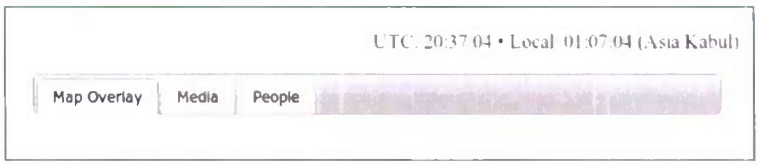


Figure 171: The timezone information for Kandahar.

9 Phase II Extension Proposal - GeoLinker

The Army's *Commercialization Pilot Program* is meant to facilitate the transition from Phase II *SBIR* contract work to acquisition by the United States Army. Additionally, it is meant to increase commercialization aspects of *SBIR* software. From their website:

> The objective of this effort is to increase Army SBIR technology transition and commercialization success thereby accelerating the fielding of capabilities to Soldiers and to benefit the nation through stimulated technological innovation, improved manufacturing capability, and increased competition, productivity, and economic growth.

Cazoodle has submitted one previous *CPP* proposal, which will help by virtue of increased clarity and completeness of our new proposal. Additionally, we hope that the growth of the *Capstone* system will demonstrate our skill in the research and development of geospatial applications.

While the *CPP* is meant to aid in acquisition and commercialization, the *SBIR* Phase II Extension is meant to allow further study of a specific problem, as identified in the Phase II contract work. Throughout the development of the *Capstone* system, we have faced the problem of linking geospatial entities across different datasets. We developed several custom solutions, *e.g.*, to link populated place features across disparate gazetteers or to link sub-city features within a given populated place.

As an extension for this Phase II work, we hope to develop a more general geospatial semantic linking software, which will work on any domain with a minimal amount of configuration.

9.1 GeoLinker Prototype

In order to elicit additional support from the United States Army for our *CPP* and *SBIR* proposals, we built a proof-of-concept prototype software, *GeoLinker*. We built this prototype to test semantic linking techniques in the geospatial feature domain of "dams."

There exist two disparate gazetteers of dam features in the United States: the *National Inventory of Dams* (NID), maintained by the US Army Corps of Engineers, while a completely separate list is maintained by the *U.S. Board on Geographic Names* (GNIS). The *NID* and *GNIS* gazetteers have not been synchronized since the early 1980s, which poses problems in inter-department collaboration and accurate inventory examination. Currently, the *NID* contains 83,988 dam features, while the *GNIS* contains only 56,940. The need for synchronization is obvious.

The *GeoLinker* prototype makes use of location-based and text-based matching at the core of its semantic linking algorithm. Location-based matching defines a factor of closeness based on geospatial location – that is, how physically close two dams are, based on their latitude and longitude. Text-based matching uses several tokenization and "fuzzy-matching" techniques to define a factor of closeness based on dam names – that is, how closely two dam's names match each other. For this matching, we chose the *GNIS* features as the "base" dams and the *NID* features as the "candidate" dams. That is, we start with a given *GNIS* feature and attempt to match one or more *NID* features to it.

Since text-based matching can become quite expensive, we first choose only a subset of candidate dams that represent probable matches. These candidates are ones which are either physically close to the base dam – that is, if we are matching dams in Illinois, it is unlikely that dams in California will provide any matches – or have their latitude and longitude undefined. We include these "undefined" examples; as a percentage of *NID* dams are missing their geospatial attributes, but should still be considered for linking. Once this selection is done, we run a set of text-based matching algorithms which produce different linking metrics.

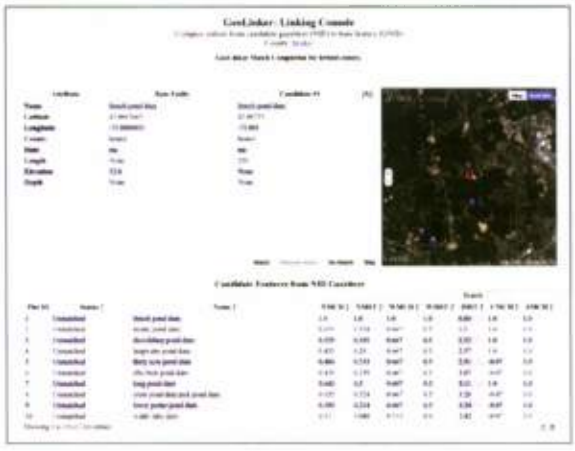


Figure 172: The operator console for the *GeoLinker* prototype.

Figure 173: The workload browser for the *GeoLinker* prototype. Massachusetts is highlighted.

Finally, as shown in Figure 172, we present the base and candidate dams in an operator console to allow an analyst to make final decisions. Throughout the *GeoEngine* project, we have found that a combination of algorithmic scoring and human finalization yield the highest rates of precision. This console allows the analyst to view our algorithms' output metrics, as well as geospatial and textual data about the proposed candidate matches. The console then allows the analyst to select one or more of these candidates to link to the base feature.

We hope that this prototype highlights *Cazoodle*'s research and development capabilities while still underlining the need for additional work in this area.

9.2 Prototype Enhancements

In order to strengthen the impact of our *GeoLinker* prototype, we made several usability improvements to our operation console. Figures 173 and 174 highlight these improvements.

The first, dubbed "workload browsing," gives the operator a graphical overview of the overall progress of gazetteer linking. Dams matching is partitioned by county so as to facilitate a sense of completion; that is, smaller workload chunks give the analyst a good set of goals and/or stopping-places. Each county is outlined on the map, with its color providing a graphic view of the linking progress therein. Red counties are wholly unmatched while green counties are wholly matched; the color fades from the former to the latter as additional links are added.

The second is a graphical progress breakdown from within the linking console itself. The progress bar is

Figure 174: The progress bar, highlighted in red.

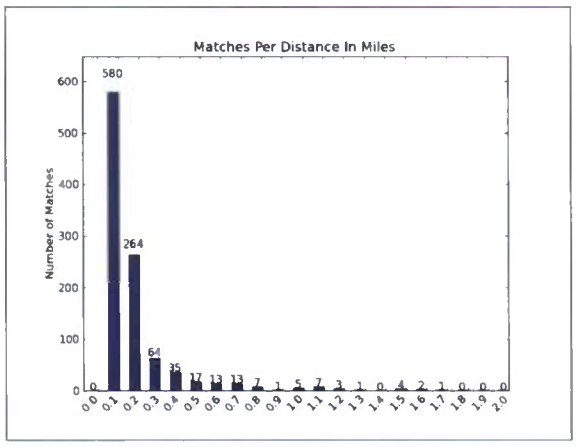


Figure 175: A graph that compares the number of linked features to their calculated distance from one another.

separated into three sections: red denotes the number of base features that have not yet been examined, blue denotes the number of base features for which the analyst has explicitly decided there is no candidate match, and green denotes the number of base features that have been assigned one or more candidate matches. The purpose of this progress bar is to give the analyst a real-time view of the amount of work they have finished and to provide a sense of completion as they work through the linking task.

9.3 Results

Our Contract Officer's Representative, Mr. Caldwell, graciously agreed to test the *GeoLinker* prototype. He focused on the state of Massachusetts, examining a total of 1,191 base dams from the *Geographic Board of Names* dataset. He matched these against the 1,602 candidate features provided by the *National Inventory of Dams*.

Of the 1,191 base features, 1,023 were matched with candidate dams. Additionally, we found that there may exist some duplicate entries in the base and candidate datasets. By examining the relationships between base and candidate dams, we determined that 1,001 of these matches were unique, and the other 22 were duplicates of base features.

Of these matches, 14 base features were matched with two candidate dams and only 3 base features were matched with three candidate dams. This data points to a small amount of duplication in the candidate, *NID* dataset as well. Additionally, 190 of the base features had no matching dam feature in the candidate dataset. Of the 1,602 candidate features, 614 were left without matches. These dam features are likely those that are unique

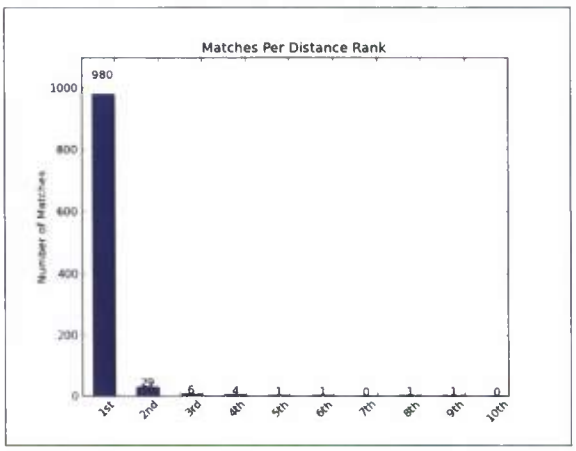


Figure 176: A graph that compares the number of linked features to their initial ranking in our candidate table.

to each data-set.

Finally, Figure 175 and Figure 176 plot the frequency of matches based on their geospatial distance and ranking in the initial candidate list, respectively. From these two figures, we see that overwhelmingly, links are most often found within 0.1 miles and as the first feature in our candidate list, but that there are still some links that do not meet these criteria. This means that, while we may be able to automatically match a large portion of the dam features, human verification is still necessary.

10 Concluding Information

In this final section we will discuss two primary topics: possible future directions for the research started in this SBIR Phase II project and our overarching conclusions on the GeoEngine project.

10.1 Future Work

The future research section is organized to reflect the major research areas that have been described in this report. These additional avenues of research are inspired by the questions and ideas that were generated as we concluded each area.

10.1.1 Hospital Discovery

Research questions generated by the Hospital Discovery task have been answered in later Phase II tasks: additional work in text extraction has been done in the Mosque Extraction and Wikipedia tasks, GeoMerging has been examined in the Capstone and GeoLinker systems, and several of our projects have included advanced operational consoles. Since the Hospital Discovery task was meant to resolve loose-ends from the SBIR Phase I work, we find no novel areas of research from this task.

10.1.2 Mosque Feature Discovery

There remain several unexplored research areas in the Mosque Feature Discovery task, both of which revolve around language. The first is an improvement in foreign language extraction. While our initial research showed that the then-current state-of-the-art was insufficiently advanced to support language parsing in Arabic areas, many advances in automated translation have since been made. Additionally, the extension to other languages was untested – it may be that our text-based extractors could be extended to Spanish or French without the same degree of difficulty, which would allow the GeoEngine system to operate in additional localities.

Second, the incorporation of Natural Language Parsing (NLP) with the text-based extractor has not been fully explored. NLP techniques can be used to give semantic annotation to each token in the input text, which can then be used as input for the rules, which carry out final entity extraction. Additionally, NLP techniques can yield

additional in-context information about otherwise one-dimensional entities: that is, various descriptive words such as "famous," "ancient" or "rarely used" might be associated as additional attributes of entity extraction.

10.1.3 Discovery of Features from Geo-tagged Images

While several of the algorithms that were developed for this task have been extended for use in the Capstone system, the full generation of a feature gazetteer has not been tested with our *Media Aggregation Engine*. Additionally, the concepts of grouping and inferencing have not been fully matured.

There is room for research in feature generation via untapped media sources and with full grouping and inferencing capabilities: that is, a system that not only aggregates media items, but uses their tags and geo-location to construct fully-formed features.

10.1.4 Mountain Attribute Discovery

One area of research left to explore was that of using interpolated data as an error-checking mechanism for manually entered values. In the initial Mountain Attribute Discovery task, we found that there were several mountains in the *NGA* data whose elevation was roughly three times that of the interpolated results. We initially postulated that this may be caused by incorrect units; that is, the value was entered as "meters" instead of "feet."

It is possible that the interpolation method missed certain mountain peak values – that is, it is reasonable for some mountains to have much higher peak elevations than the interpolated results – but there are also many areas where checking the manually entered value against an expected or interpolated value would be helpful in providing error checking and accuracy feedback.

10.1.5 Neighborhood Extraction

There are several areas for further improvement in the neighborhood task. We only briefly experimented with the interaction between different clustering algorithms; it may be possible to find more suitable clustering algorithms for working in tandem to achieve higher results. It may also be possible to tailor the operations on a neighborhood feature set to the general shape of the natural set, that is, if it contains a tight cluster and many outliers, it may be beneficial to run the "Circle Cluster" algorithm, whereas if the neighborhood set is initially looser, it may be better to use the "Median Cluster" algorithm.

Additionally, questions regarding neighborhood generation in foreign localities may be pursued: What geospatial feature types make for the best boundary generation? What are the differences in boundary generation in various geographies? How can these differences be addressed programmatically?

10.1.6 Wikipedia Traversal

The Wikipedia Traversal task yields several intriguing areas for further research. First, the extension of gazetteer generation to additional feature types. In our Phase II work, the GeoEngine project focused mainly on geospatial features: mountains, lakes and populated places. Can these extraction techniques be extended to other features?

Second, the Wikipedia autocorrection task only attempted to identify missing attributes for "populated place" features. The techniques used for attribute aggregation and missing attribute identification may be extended to other sets of Wikipedia data. Additionally, research into the types of missing attributes may answer questions regarding the way in which a human editor interacts with Wikipedia.

10.1.7 Capstone System

There are many directions for future research that spawn from the Afghanistan Places Profile Capstone task. The most obvious is an extension to additional geographies: What types of challenges might be faced in adding additional countries? What are the key similarities in "Profile" generation from one locality to another, and how might these be used to automate the extension process?

Specific modules of the Capstone system might provide a platform for additional research: Can context-specific information be extracted from a linked Wikipedia article? How do the trends of each media provider, *i.e.*, *Flickr*, *Twitter* or news sources, compare? How can sub-city features be semantically linked in order to provide a more comprehensive ground-view of a populated place? What additional data can be extracted from the relationship between specific people and populated places or other people?

10.1.8 GeoLinker

The GeoLinker prototype may be extended in many ways. The first is in rapid domain expansion – that is, methods for extending geospatial semantic linking into any general domain must be researched. Secondly, the machine-assisted linking algorithms require additional research in order to create a machine-learning architecture that responds to the analysts' final matching decisions. Finally, operational console designs must be researched so that the final design motivates the analyst to complete tasks; that is, linking tasks should be presented in such a way as to make the work productive and enjoyable.

10.2 Conclusions

Through the pursuit of each of the GeoEngine tasks, we believe that we have made several important contributions to state-of-the-art geospatial research. Key highlights are as follows:

- We have shown the usefulness of true "Deep Web" search in the extraction of geospatial data from Web sources over traditional "Surface Web" searches, which tend to yield not only less data, but unstructured entities that lack context
- We have shown the ability of aggregate data to overcome limitations of single datasets in the areas of gazetteer creation and feature extraction
- We have shown the value of geography as an organizing principle – that geospatial proximity is helpful not only as regards semantic linking, but in confidence-increasing feature grouping
- We have shown the importance of concordance mapping for linked data; where a single dataset may be lacking valuable attributes, the network created by a linked concordance may recall these values from another source
- We have shown the usefulness of emerging "No-SQL" technology in the geospatial sphere by virtue of its organizational use in our Capstone and GeoLinker systems
- Finally, we have shown the increased usefulness of combined human-machine consoles – that is, where results have been refined by machine processes but also confirmed by human analysts.

References

[1] Stanford arabic parser. http://nlp.stanford.edu/software/parser-arabic-faq.shtml.

[2] http://apartments.cazoodle.com/search/index.php?q=Chicago,+IL&beds=-1&baths=-1&price_min=-1&price_max=-1.

[3] http://chicagomap.zolk.com/.

[4] http://code.flickr.com/blog/2008/10/30/the-shape-of-alpha/.

[5] http://code.flickr.com/blog/2009/04/07/the-only-question-left-is/.

[6] http://code.google.com/apis/ajaxsearch/documentation/.

[7] http://code.google.com/apis/maps/documentation/geocoding/.

[8] http://code.google.com/apis/youtube/overview.html.

[9] http://developer.yahoo.com/search/local/V3/localSearch.html.

[10] http://english.aljazeera.net/indepth/features/2011/09/20119592211411357.html.

[11] http://english.aljazeera.net/news/asia/2011/08/201182175746112785.html.

[12] http://eu.techcrunch.com/2009/01/09/house-is-on-fire-were-out-shit-twitter-proves-itself-again/.

[13] http://matadornetwork.com/change/how-to-follow-the-egyptian-uprising-on-twitter.

[14] http://news.cnet.com/8301-13577_3-20031600-36.html.

[15] http://opencv.willowgarage.com/wiki/.

[16] http://polygon.origo.ethz.ch/.

[17] https://picasaweb.google.com/home.

[18] http://strategy.wikimedia.org/wiki/Proposal:Implement_OAuth_for_MediaWiki_(and_employ_in_Wikimedia).

[19] http://twitter.com/.

[20] http://wikimapia.org/.

[21] http://www.ams.org/mathscinet/collaborationDistance.html.

[22] http://www.apartments.com/Results.aspx?page=results&stype=city&city=chicago&state=il&prvpg=5&Rent_Minimum=0&Rent_Maximum=99999.

[23] http://www.boorah.com/restaurants/best-of/5463/IL/Chicago.html?start=0.

[24] http://www.businessinsider.com/2009/1/us-airways-crash-rescue-picture-citizen-jouralism-twitter-at-w

[25] http://www.flickr.com/services/.

[26] http://www.geometrylab.de/RandomPolygon/index.html.en#twopeasants.

[27] http://www.google.com/earth/index.html.

[28] http://www.mediabistro.com/fishbowlny/media_events/twittering_mumbai_citizen_journalism_gets_one_step_closer_to_the_mainstream_101970.asp.

[29] http://www.mediawiki.org/wiki/API:Main_page.

[30] http://www.miislita.com/searchito/levenshtein-edit-distance.html.

[31] http://www.mongodb.org/.

[32] http://www.move.com/apartmentsforrent-search/chicago_il.

[33] http://www.msnbc.msn.com/id/44453535/ns/world_news/t/clinton-al-qaeda-behind-unconfirmed-threat-us.

[34] http://www.mynewplace.com/search?back=T&q=chicago,+il.

[35] http://www.reportr.net/2011/02/16/visualization-egyptian-uprising-twitter/.

[36] http://www.techcrunch.com/2009/01/08/breakingnewson-from-twitter-account-to-public-news-wire-service

[37] http://www.techcrunch.com/2009/06/17/is-twitter-the-cnn-of-the-new-media-generation/.

[38] http://www.techcrunch.com/2009/08/20/twitter-can-now-know-where-you-tweet/.

[39] http://www.topix.com/.

[40] Hinneburg A. and Keim D. A. An efficient approach to clustering in large multimedia databases with noise. *KDD*, 1998.

[41] Steven Abney. Parsing by chunks. In *Berwick, Abney, and Tenny, editors, Principle-Based Parsing.* Kluwer Academic Publishers, 1991.

[42] Einat Amitay, Nadav Har'El, Ron Sivan, and Aya Soffer. Web-a-where: geotagging web content. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 273–280. ACM, 2004.

[43] Chris Anderson. The long tail. In *Wired*. Wired.com, October 2004.

[44] Global Administrative Areas. *GADM* database of global administrative areas. 2010.

[45] Yassine Benajiba, Mona Diab, and Paolo Rosso. Arabic named entity recognition using optimized feature sets. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 284–293, Honolulu, Hawaii, October 2008. Association for Computational Linguistics.

[46] Yassine Benajiba and Paolo Rosso. Arabic named entity recognition using conditional random fields. In *Proceedings of the Workshop on HLT and NLP within the Arabic World*, 2008.

[47] GEOnet Names Server NGA Bethesda. National geospatial-intelligence agency. "country files (gns).". NGA GEOnet Names Server (GNS), 2010.

[48] Didier Bourigault. Surface grammatical analysis for the extraction of terminological noun phrases. In *Proceedings of the 14th conference on Computational linguistics*, pages 977–981. Association for Computational Linguistics, 1992.

[49] Thorsten Brants. Tnt - a statistical part-of-speech tagger. *CoRR*, cs.CL/0003055, 2000.

[50] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. *ACM SIGMOD*, 2000.

[51] Eric David Brill. *A corpus-based approach to language learning*. PhD thesis, Philadelphia, PA, USA, 1993.

[52] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. Extracting content structure for web pages based on visual representation. In *Fifth Asia Pacific Web Conference (APWeb2003)*, 2003.

[53] Deepayan Chakrabarti, Ravi Kumar, and Kunal Punera. Page-level template detection via isotonic smoothing. *Proceedings of 16th international conference on World Wide Web*, 2007.

[54] Soumen Chakrabarti, Kunal Puncra, and Mallela Subramanyam. Accelerated focused crawling through online relevance feedback. In *WWW*, pages 148–159, 2002.

[55] Tao Cheng and Kevin Chen-Chuan Chang. Entity search engine: Towards agile best-effort information integration over the web. In *Proceedings of the Third Conference on Innovative Data Systems Research (CIDR 2007)*, pages 108–113, Asilomar, Ca., January 2007. Extended System Demo Description.

[56] Tao Cheng, Xifeng Yan, and Kevin Chen-Chuan Chang. EntityRank: Searching entities directly and holistically. In *Proceedings of the 33rd Very Large Data Bases Conference (VLDB 2007)*, Vienna, Austria, September 2007.

[57] Tao Cheng, Xifeng Yang, and Kevin Chen-Chuan Chang. Supporting entity search: a large-scale prototype search engine. In *Proceedings of the 2007 ACM SIGMOD Conference (SIGMOD 2007)*, pages 1144–1146, Beijing, China, June 2007.

[58] Junghoo Cho, Hector Garcia-Molina, and Lawrence Page. Efficient crawling through url ordering. *Computer Networks*, 30(1-7):161–172, 1998.

[59] Kenneth Ward Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the second conference on Applied natural language processing*, pages 136–143. Association for Computational Linguistics, 1988.

[60] Paul Clough. Extracting metadata for spatially-aware information retrieval on the internet. In *GIR '05: Proceedings of the 2005 workshop on Geographic information retrieval*, pages 25–30. ACM, 2005.

[61] M. Collins and Y. Singer. Unsupervised models for named entity classification. *Proceedings of the Joint SIGDAT Conference on EMNLP and VLC*, 1999.

[62] Backstrom L. Huttenlocher D. Crandall, D. J. and J. Kleinberg. Mapping the world's photos. In *n WWW '09:* Proceedings of the 18th international conference on World wide web, page 761–770. ACM, 2009.

[63] Martin Doerr and Manos Papagelis. A method for estimating the precision of placename matching. *IEEE Trans. on Knowl. and Data Eng.*, 19(8):1089–1101, 2007.

[64] David I. Donato. Fast, inclusive searches for geographic names using digraphs. In *U.S. Geological Survey Techniques and Methods, Book 7*, chapter A1. 2008.

[65] *PEAKLIST*. Afghanistan and central/southern pakistan. 2010.

[66] *Wikipedia: The Free Encyclopedia*. `http://en.wikipedia.org/wiki/Category:Mountains_of_Afghanistan`. 2010.

[67] *Wikipedia: The Free Encyclopedia*. `http://en.wikipedia.org/wiki/Inverse_distance_weighting`. 2010.

[68] CGIAR Consortium for Spatial Information. Srtm 90m digital elevation data. SRTM 90m Digital Elevation Data, 2004.

[69] Sheikholeslami G., Chatterjee S., and Zhang A. Wavecluster: A multi-resolution clustering approach for very large spatial databases. *VLDB*, 1998.

[70] GeoNames. `http://www.geonames.org/export/web-services.html`. GeoNames WebServices, 2010.

[71] Afghanistan GeoNames Advanced Search and Feature Class T. `http://www.geonames.org/statistics/afghanistan.html`. 2010.

[72] Hany Hassan and Jeffrey Sorensen. An integrated approach for Arabic-English named entity translation. In *Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages*, pages 87–93, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[73] Alexander G. Hauptmann and Andreas M. Olligschlaeger. Using location information from speech recognition of television news broadcasts. In *Proceedings of the ESCA workshop: Accessing information in spoken audio*, pages 102–106. Cambridge University, April 1999.

[74] Edwin M. Knox and Raymond T. Ng. Algorithms for mining distance-based outliers in large datasets. *VLDB*, 1998.

[75] Julian Kupiec. An algorithm for finding noun phrase correspondences in bilingual corpora. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 17–22. Association for Computational Linguistics, 1993.

[76] Jochen L. Leidner. Toponym resolution in text: annotation, evaluation and applications of spatial grounding. *SIGIR Forum*, 41(2):124–126, 2007.

[77] Jochen Lothar Leidner. *Toponym Resolution in Text*. PhD thesis, University of Edinburgh, jun 2007.

[78] Huifeng Li, Rohini K. Srihari, Cheng Niu, and Wei Li. Location normalization for information extraction. In *Proceedings of the 19th international conference on Computational linguistics*, pages 1–7, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

[79] Huifeng Li, Rohini K. Srihari, Cheng Niu, and Wei Li. Infoxtract location normalization: a hybrid approach to geographic references in information extraction. In *Proceedings of the HLT-NAACL 2003 workshop on Analysis of geographic references*, pages 39–44. Association for Computational Linguistics, 2003.

[80] Ester M., Kriegel H.-P., Sander J., and Xu X. Density-based algorithm for discovering clusters in large spatial databases with noise. *KDD*, 1996.

[81] Lewis Mumford. The neighbourhood and the neighbourhood unit. *Town Planning Review*, 24, 1954.

[82] Simon E. Overell and Stefan M. Rüger. Identifying and grounding descriptions of places. In *GIR*, 2006.

[83] Agrawal R., Gehrke J., Gunopulos D., and Raghavan P. Automatic subspace clustering of high dimensional data for data mining applications. *ACM SIGMOD*, 1998.

[84] Lance A. Ramshaw and Mitchell P. Marcus. Text chunking using transformation-based learning. *CoRR*, cmp-lg/9505040, 1995.

[85] Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. *EMNLP 1*, 1996.

[86] Erik Rauch, Michael Bukatin, and Kenneth Baker. A confidence-based framework for disambiguating geographic terms. In *Proceedings of the HLT-NAACL 2003 workshop on Analysis of geographic references*, pages 50–54. Association for Computational Linguistics, 2003.

[87] R. Nair S. Ahern, M. Naaman and J. Yang. World explorer: Visualizing aggregate data from unstructured text in geo-referenced collections. In *Proceedings of the Seventh ACM/IEEE-CS Joint Conference on Digital Libraries, (JCDL 07)*, 2007.

[88] D. Samy, A. Moreno-Sandoval, and J. M. Guirao. A proposal for an arabic named entity tagger leveraging a parallel corpus (spanish-arabic). In *Proceedings of International Conference on Recent Advances on Natural Language Processing RANLP*, Borovets, Bulgaria, 2005.

[89] David A. Smith and Gregory Crane. Disambiguating geographic names in a historical digital library. In *ECDL '01: Proceedings of the 5th European Conference on Research and Advanced Technology for Digital Libraries*, pages 127–136. Springer-Verlag, 2001.

[90] Soft Surfer. Fast winding number inclusion of a point in a polygon by dan sunday. 2006.

[91] Fawcett T. and Provost F. Adaptive fraud detection. *Data Mining and Knowledge Discovery Journal, Kluwer Academic Publishers*, 1997.

[92] Ng R. T. and Han J. Efficient and effective clustering methods for spatial data mining. *VLDB*, 1994.

[93] Zhang T., Ramakrishnan R., and Linvy M. Birch: An efficient data clustering method for very large databases. *ACM SIGMOD*, 1996.

[94] N. Good T. Rattenbury and M. Naaman. Towards automatic extraction of event and place semantics from flickr tags. In *In* Proceedings of the Thirtieth International ACM SIGIR Conference, (SIGIR 07), *year = 2007,*.

[95] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 173–180. Association for Computational Linguistics, 2003.

[96] Jeffrey Travers, Stanley Milgram, Jeffrey Travers, and Stanley Milgram. An experimental study of the small world problem. *Sociometry*, 32:425–443, 1969.

[97] Barnett V. and Lewis T. Outliers in statistical data. *John Wiley*, 1994.

[98] Atro Voutilainen. Nptool, a detector of english noun phrases. In *Proceedings of the Workshop on Very Large Corpora*, pages 48–57. Association for Computational Linguistics, 1993.

[99] DuMouchel W. and Schonlau M. A fast computer intrusion detection algorithm based on hypothesis testing of command transition probabilities. *KDD*, 1998.

[100] Wang W., Yang J., and Muntz R. Sting: A statistical information grid approach to spatial data mining. *VLDB*, 1997.

[101] Allison Gyle Woodruff and Christian Plaunt. Gipsy: automated geographic indexing of text documents. *J. Am. Soc. Inf. Sci.*, 45(9):645–655, 1994.

[102] G. K. Zipf. *Human Behavior and the Principle of Least Effort.* Addison-Wesley, Cambridge, Massachusetts, 1949.